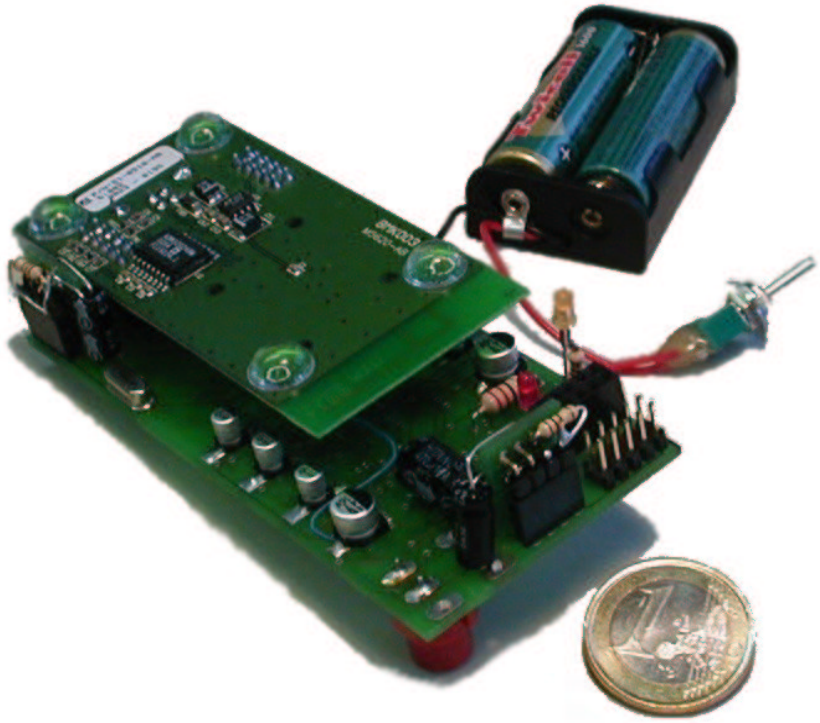
BLUEWAND

A versatile remote control and pointing device

University of Karlsruhe, Germany



Team members Manuel Odendahl
Markus Klein
Christian Ceelen
Alexander Lange

Mentor Dr. Thomas Fuhrmann

University of Karlsruhe

1 Abstract

BLUEWAND is a small device that can be used to control other Bluetooth enabled devices by hand-movements. It can be used with a mobile phone or a DiscMan, during a beamer presentation, or to play games while being on the move. BLUEWAND is also suited for controlling machines on the factory floor and supporting disabled people.

Technically, BLUEWAND is based on a 4-axis accelerometer system combined with specific analysis algorithms that detect hand-movements and extract both movement and pointing direction. It employs two Analog Devices ADXL202E sensors and an Atmel AVR ATmega103 microcontroller. We have built a complete working prototype and integrated it into a common operating system, enabling various sample applications to use the BLUEWAND.

2 System overview

2.1 Vision

Bluetooth technology is promoting the trend of the vanishing computer. Most obviously, Bluetooth helps to break up universal machines into separate functional entities, e.g., earphone, mobile phone, and PDA. As a result, the electronic world decomposes into highly specialized tools. Given this vison, there is an immediate need for a specialized control device that acts as human computer interface (HCI) to all these devices.

2.2 Objectives

BLUEWAND is designed to provide a HCI for situations where other forms of HCI would be unnatural. Typical examples that guided our design process are:

- Audio applications consisting of an earphone and e.g. a mobile phone or DiscMan where both involved devices are an awkward place for the HCI since one is plugged into your ear and the other stored safely in your clothing or in a backpack.
- Beamer presentations where the ease of using a laser-pointer is still unparalleled by any digital control device.
- Mobile gaming of the future where people use head-mounted devices as display and BLUEWAND e.g. as laser-saber.

During the design process it became clear that the BLUEWAND might also be useful for other purposes besides the major use-cases that guided our design: Directing robots and other machines on the factory-floor or construction sites by demonstrating the desired movements; Providing authentification for access control by a combination of cryptographic key stored in the device with a personal signature-movement that protects against loss or theft of the BLUEWAND; Guiding blind people in unacquainted areas. In order to ease a future universal usage, BLUEWAND should also be able to replace various devices like remote controls, joysticks, etc.

2.3 Team organization and design methodology

After defining our objectives, we cut the project into different tasks, each of which was assigned to a team member who became responsible for that task. These tasks were: sensor

system, hardware (excluding sensors), Bluetooth stack, software aspects (excluding BT stack and applications), and the two different application scenarios ”controlling” (DiscMan, mobile phone, etc.) and ”pointing” (mouse, joystick, etc.). Depending on individual work-load from lectures and other university activities some team members were concerned with more than one task.

Collaboration within the team was ensured by regular meetings of the whole team, where interfaces between the different tasks were discussed and decided. Additionally, these meetings gave us the opportunity to help one another with problems that occured within the individual work packages.

Due to the different nature of the tasks, we also pursued different design methodologies. They are more closely described in section 3. To share specifications, source code, documentation, and other project material, we used the Concurrent Versions System (CVS).

2.4 Performance requirements

To meet the objectives listed above, we designed BLUEWAND to comfortably fit into the human hand. This limited not only size but also weight of the device. Both requirements then influenced most of the design decisions (power supply, power consumption, sensor technology, etc.) as will be discussed in the following. Since BLUEWAND aims at the mass market, we paid special attention to cost and manufacturability issues.

2.5 Innovation and Related Technology

Data gloves are a well known tool for digitization of hand-movements. However, we think that permanently wearing a glove is a handicap in practice, keeping users from using it widely. BLUEWAND is a cost-effective tool that focuses on pointing and controlling. Thus the overhead of a full data glove can be avoided.

We know of two other projects that pursue the same approach, yet are based on different measurement principles:

- IBM has shown a prototype of a wrist-watch named WatchPad that contains a 2-axis accelerometer. Since a watch can be worn permanently without blocking one's hands, this is a very good place for this device. However, such a solution is only capable of detecting arm movements not the much more common hand movements. Even more, BLUEWAND is based on a 4-axis system yielding the full spatial orientation.

- Gyration offers a pointing device (GyroMouse) that is based on a miniature gyroscope. Due to the different measurement principle, GyroMouse is better capable of detecting rotations. BLUEWAND on the other hand benefits from its absolute tilt measurement and the ability to detect both linear and rotational movements.

The current interest in employing accelerometers or gyroscopes as HCI shows in our opinion the great potential behind these ideas.

3 Implementation

3.1 Hardware

The hardware design was governed by the need for a small, lightweight and cost-efficient device. This guided our choice of microcontroller and sensors and led us to use SMD parts where possible. Some trade-offs had to be made, since our design had to house the Ericsson Bluetooth module that we were provided with (see below).

In the beginning we used a bread board and the Atmel STK300+ development kit to test our design. Timing conditions were measured using an oscilloscope. After a stable design stage was reached, the circuit was transformed into a printed circuit board by the EAGLE layout programm. Actual manufacturing was outsourced to a company specialized on that task (see cost table 1).

3.1.1 Microcontroller: Atmel AVR ATmega103

BLUEWAND is based on the Atmel AVR ATmega 103 microcontroller. This is a high-performance low-power RISC architecture with 4 kB of internal SRAM. Its 128 kB, in-system programmable flash facilitates quick development cycles. The on-chip UART provides connection to the Bluetooth module. The 16-bit timer/counter can capture interrupt timestamps needed for the sensors. In the BLUEWAND, the ATmega 103 is operated at 7.3728 MHz, i.e. 22 % above the specified value of 6 MHz but below the maximum value of the AVR family. This clock frequency matches the UART baud rate, gives us a little bit of extra computing power, and has been proven to work stably in the BLUEWAND.

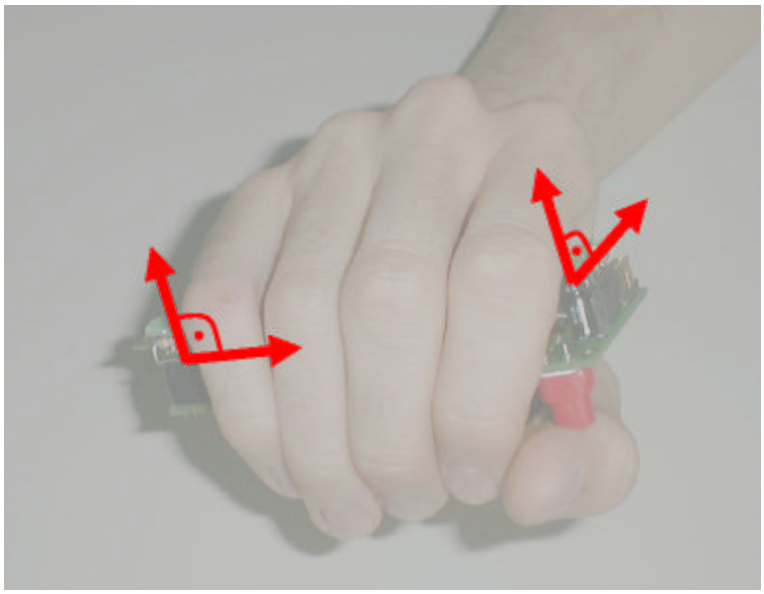


Figure 1: BLUEWAND held in the human hand – Red arrows show sensor axes

3.1.2 ADXL202 accelerometers

Movements of the BLUEWAND are tracked by two ADXL202 accelerometers from Analog Devices. These sensors are low-cost, low-power, complete 2-axis accelerometers, available as SMD packages. Compared to gyroscope approaches, these sensors allow us to measure the absolute tilt values (i.e. no drift in measurements) and the full lateral movement of the BLUEWAND. As one sensor can track two orthogonal axes, we had to use two orthogonal sensors (i.e. four axes) to track the full 3D orientation. Since the casing of the final BLUEWAND product is supposed to fit into the human hand in one defined orientation only, we can use the fourth (i.e. duplicate) axis to detect wrist rotation (cf. figure 1). To realize this specific orientation, the sensors were placed together with their external resistors and capacitors on two mini-boards mounted upright on the board via pin connectors. The complete arrangement of the sensors and their axes is also shown in figure 2. We decided to use the digital output to ensure a better signal quality. The sensors provide analog and digital output. The latter uses

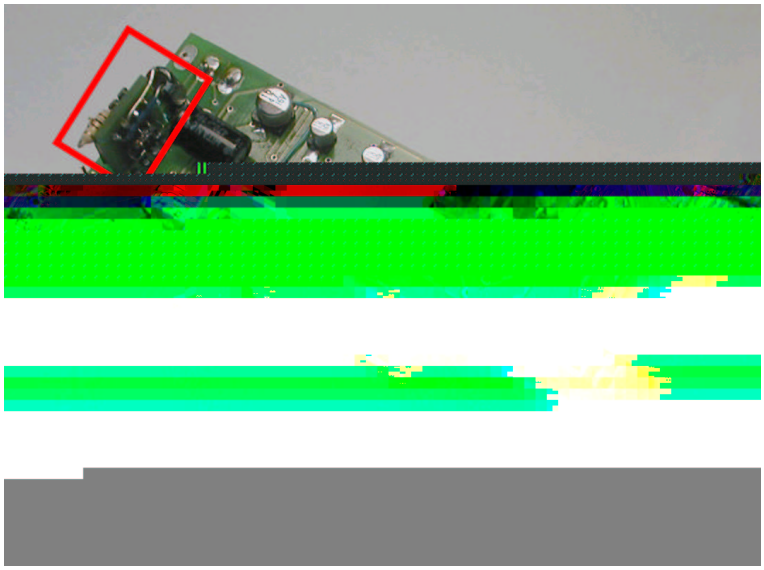


Figure 2: BLUEWAND Board with both sensor mini-boards attached (red boxes)

duty cycles that are proportional to the acceleration on each of the corresponding axes. A duty cycle is the ratio of pulse-width t_i to period T (cf. figure 3).

3.1.3 Aquisition and preprocessing of sensor output

The sensor output lines are directly connected to the I/O ports of the ATmega103. Additionally, the sensors trigger an interrupt each time one of the signals changes. This is achieved by combining the sensor signals using XOR-gates. Two monoflops, one reacting on a falling edge and the other on a rising edge, generate a pulse that is long enough for the interrupt routine to record the sensor values. This is needed to properly identify sensor state and interrupt timestamp. The outputs of the monoflops are wired through an OR-gate to the interrupt capture pin of the AVR ATmega103.

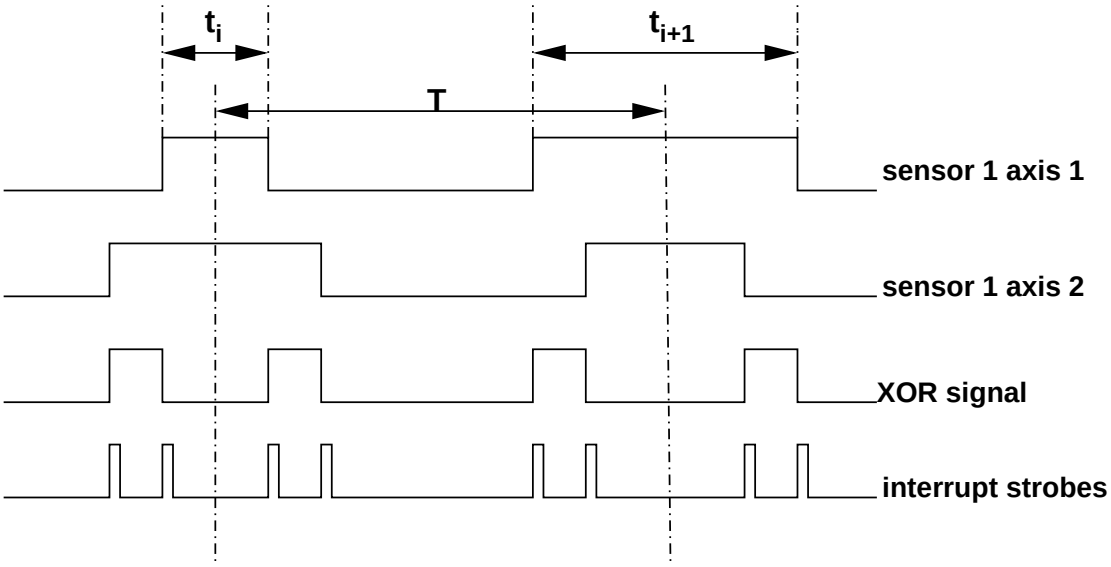


Figure 3: Duty cycles of the sensors and generation of interrupt strobes

3.1.4 Connection of Ericsson Bluetooth module

The BLUEWAND is equipped with connectors for the UART and power supply pins of the Ericsson Bluetooth module, so that the module can be put (upside-down) on the board. Additionally, the microcontroller's TxD and RxD ports are connected to a MAX232ECSE level converter. This step is only required because we didn't want to change the provided Ericsson module. In a final product, the level converter is obsolete: Then the voltage can also be changed from 5V to 3.3V, thus further reducing the power consumption.

3.1.5 Power Supply

We use a Maxim MAX1674 step-up converter as power supply for our board, producing a stabilized 5V output from two AA batteries, keeping size and weight of the device low. The achieved battery lifetime is approximately one day, which is rather acceptable. In a final product, we could even use a single AA battery to provide the minimum 0.7V required by

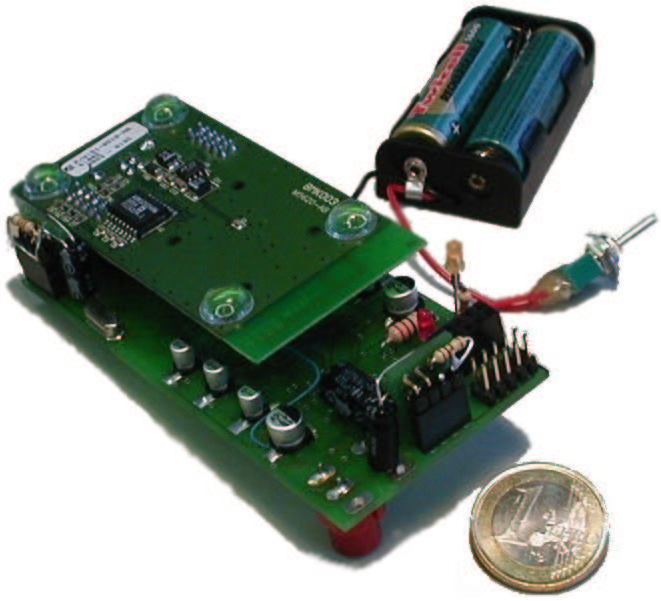


Figure 4: BLUEWAND board with mounted Ericsson bluetooth module and power supply

the MAX1674. This was not possible with our prototype, as the level converter had stability problems when running at this voltage. A final board could hence not only be considerably more power efficient, but also more lightweight.

3.1.6 Interfaces

Besides the sensors and the UART, our board has several external interfaces:

- In-circuit programming interface: The AVR ATmega103 is programmed using a simple, STK200 compatible dongle attached to the parallel port of the PC. As the TxD and RxD pins of the UART interface are also used as MISO-MOSI programming pins, an HC 4053 multiplexer enables us to program the microcontroller without hardware changes.

[1]

[1]We found an error in the official documentation of the STK300+ board. The first pin of the microcontroller (PEN) is connected to Vcc, not ground as indicated.

- Two status LEDs, indicating power and programming status, enable us to control the status of our board. Further LEDs can be attached to provide end-user feedback.

- A button is needed for point and click applications. The prototype has solder pads for a button. In the final product this button can be mounted at the casing of the board.

- For haptical feedback to the user, a small circuit is integrated on the board to control a vibration stimulator. We use a small motor with an excentrical weight that we took from a mobile phone add-on. This vibration stimulator should also be mounted at the casing of the final product.

The overall circuit schematics is shown in figure 3.1.6. The total cost of our prototype is given in table 1. The total weight of BLUEWAND is 100 g (25 g BLUEWAND board, 15 g Ericsson module, and 60 g batteries).

#	Part	costs
1	ATmega 103	22.50 $
2	ADXL202	39.90 $
1	MAX1674	1.60 $
1	MAX232	2.10 $
1	Vibration Engine	$\approx$ 8.95 $
1	Board	31.25 $
	TTL compatible glue logic, resistors, capacitors	$<$ 10 $
	$\sum$	116.30 $

Table 1: Total costs of the prototype's hardware components

3.2 Data Capture

As described above, the accelerometers are connected to the AVR ATmega103 such that edges in the sensor signals trigger an interrupt which automatically records a 16bit-timestamp. At

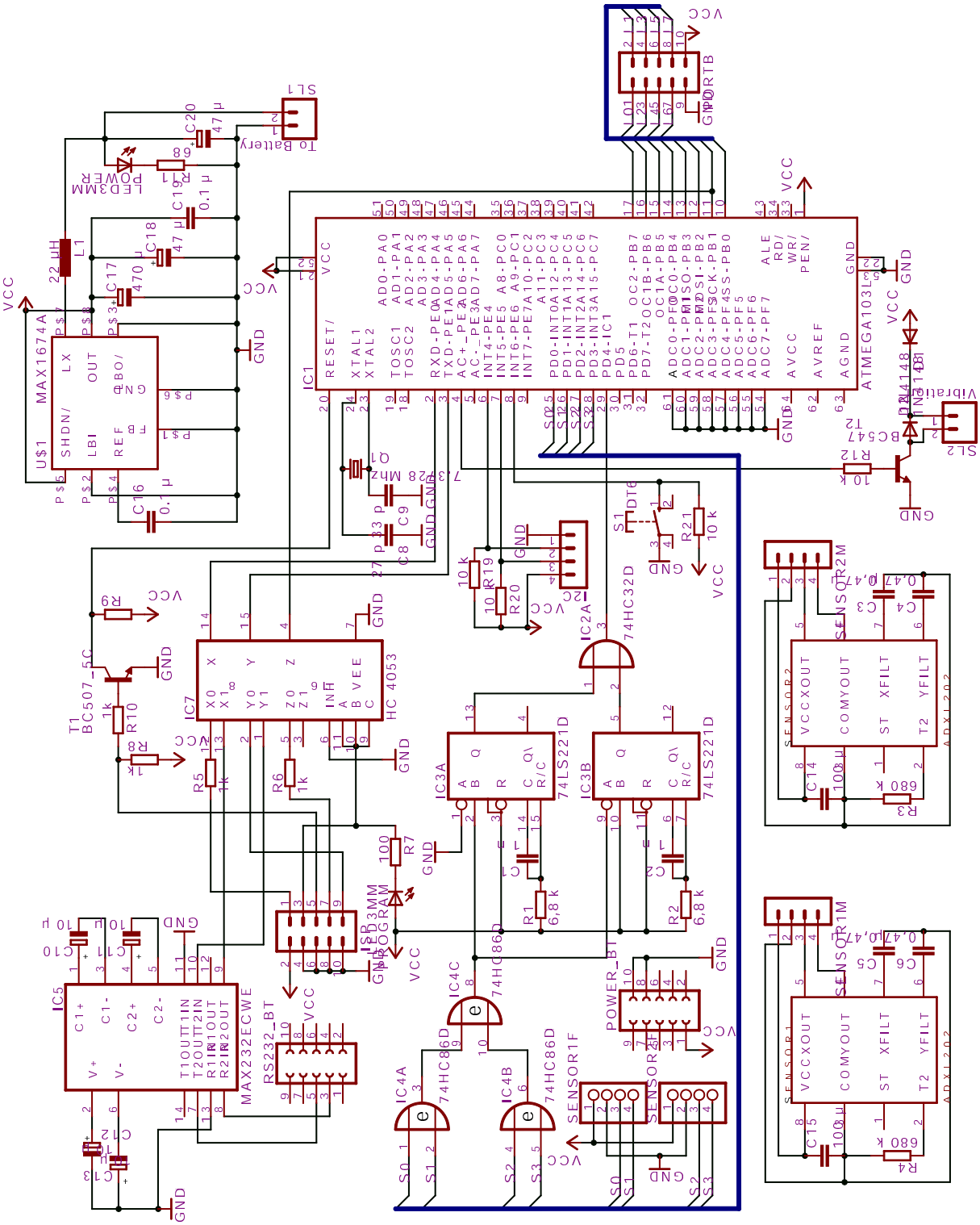


Figure 5: Circuit schematics

7.373MHz, the corresponding timer wraps every 8.89μs, so the sensors are required to run at more than 112 cycles per second. designed to work with 184 cycles and a A 10Hz low-pass filter in the accelerometers satisfies the Nyquist condition and reduces noise in the acceleration measurement. Both cycle-time and low-pass filter can be controlled by the capacitors and resistors on the sensors' mini-boards. The expected noise of well below 1% is confirmed by our results.

Upon each interrupt request the AVR ATmega103 reads the four sensor output lines and stores the pattern together with the corresponding timestamp in a circular buffer. All further preprocessing is done outside the interrupt handler in order to keep the dead time as low as possible[2]. Rising and falling edges are combined to calculate both the duty-cycle t_i and the sensor cycle T of the corresponding axis (cf. figure 3). Since T is only slowly varying, the rare case of an erroneous measurement is detected by a sanity check on T. If a value is failing this test, the corresponding axis is marked as invalid for that sample point.

3.3 Data Analysis

These raw acceleration values can either be used directly or be subjected to further preprocessing. One possible step is scaling the raw values to physical values using the gauge provided by the earth's gravitational field. The required gauge values need to be measured for all three fundamental orientations while the device is in rest. These values can be stored in the EEPROM since they only need to be measured during initial calibration of the BLUEWAND.

Another preprocessing step is the separation of tilt and motion-induced acceleration. After testing several approaches we came up with our fundamental assumption that both components

[2]The same care was applied to all other parts of the code where interrupts had to be disabled for various reasons.

can be separated due to their different timescales: Tilt is assumed to be either only slowly varying or – if changing more quickly – to change steadily to a new value without oscillation. Thus we can assume all oscillations on short timescales (less than about 300ms) to be caused by movements, i.e., accelerations followed by retardations. This separation is achieved by a least-square fit procedure with a window of 64 sample points. The fit is performed both forwards and backwards. Afterwards, both fits are combined by a weighted selection such that the resulting fit closely follows even sharp bents in the tilt curve instead of erroneously smoothing them out (cf. figure 6). This algorithm proved to be the best variant among many approaches we studied. The inevitable latency that is introduced by the 300ms fit window has shown to be pretty acceptable in practice[3]. As an additional advantage, the computing complexity of the fit algorithm does not depend on the window size, yielding a good performance.

After the separation of tilt and motion-induced acceleration the latter can be integrated to a velocity vector. Depending of the separation accuracy, this integration may or may not be sufficiently exact to match $v = 0$ after a movement. To avoid drifts by erroneous velocity residuals, our analysis algorithm resets the velocity to $v = 0$ if the acceleration change falls below a certain threshold (cf. figure 7).

3.4 Bluetooth stack

Our Bluetooth [1] software operates within two environments that are bridged via the Bluetooth link: firstly, sensor data that is gathered and preprocessed on the BLUEWAND, secondly, it has to be further processed on the controlled device. The BLUEWAND software needs to run on an AVR ATmega103 microcontroller, i.e. it has to work with less than 4 kB of RAM

[3] This result might be surprising if one knows that the human's auditive cortex is sensitive to latencies as small as 100ms. However, haptical feedback seems to be less rigorous.

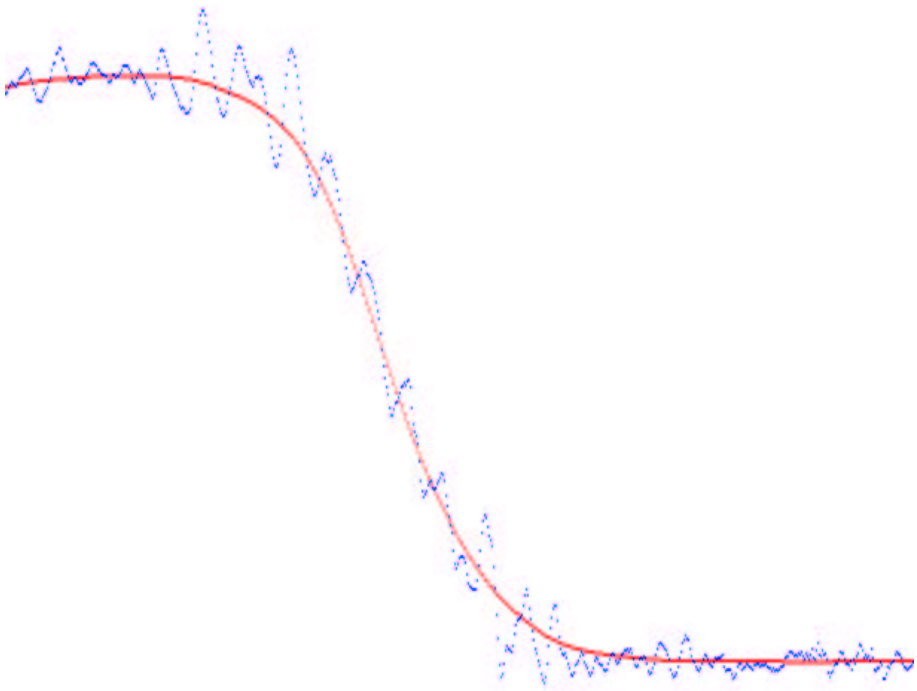


Figure 6: Single axis showing tilt-like hand-movement. Raw data (blue) shows saccadic movement that is typical for human motion. Smoothed data (red) contains undistorted tilt.

and 128 kB of flash memory, while our sample applications are implemented on a PC running Linux. Since up to now no flexible and free Bluetooth stack for the AVR ATmega103 has been available, we decided to write our own. Things looked different on the Linux PC: the BlueZ Bluetooth stack is integrated into the linux kernel since version 2.4, Axis Communication has opensourced its version of the OpenBT stack and IBM has made its BlueDrekar stack available. But since even the ”official” stack (BlueZ) has stability problems, and since we needed a stack for the AVR ATmega103 anyway, we decided to combine the AVR and Linux Bluetooth stacks to make a single, flexible and portable stack.

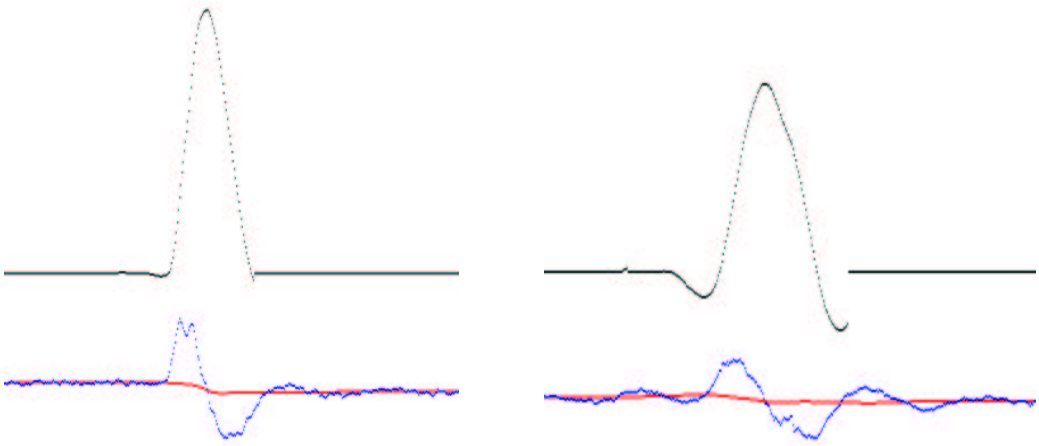


Figure 7: Single axis showing thrust-like hand-movement. Lower curves show raw and smoothed data. Upper curves (black) show integrated data, i.e. velocity values. Note that an unsteady background and motion timescale above 300ms *(right figure)* can lead to residuals that the algorithm automatically resets as soon as the movement stops.

3.4.1 Development environment

Thus, we had to write a software which would be able, with a few exceptions, to compile for both a Linux host computer and an AVR ATmega103 microcontroller. The stack design was separated into platform dependent (UART communication, device initialization and debugging/error output) and platform independent (HCI implementation, Bluetooth stack core) routines and modules.

Crossdevelopment was quite easy, as the GNU toolchain (containing the GNU C compiler, assembler, linker, object dumper), a conforming freestanding implementation of the standard C library (that is, a compliant subset of the ANSI C library) and a flash program to upload the software were available for the AVR microcontroller family. With these opensource, UNIX based tools, we were able to write platform independent code. Special attention was given to portability issues on the host side: the stack conforms to the Single Unix Specification [2], and care was given to byte order and memory alignment issues, so the software would also work on

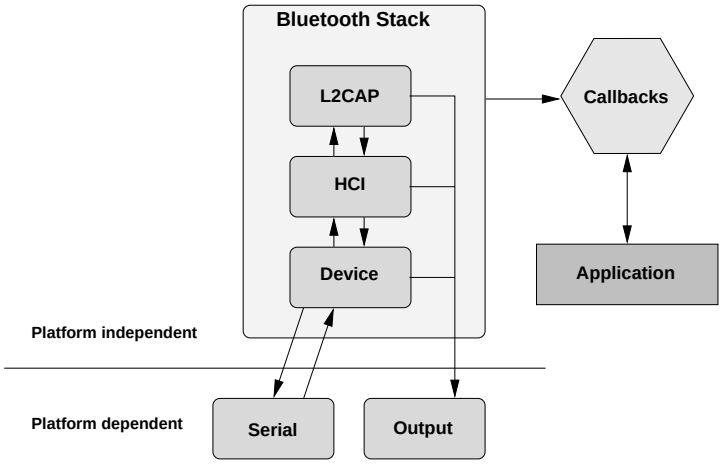


Figure 8: Internal organization of the stack

other UNIX flavors and hardware architectures (our Bluetooth stack was tested on Linux/x86, Linux/Sparc, OpenBSD/Sparc, HP-UX/PaRISC and tru64/alpha).

3.4.2 Stack design

The Bluetooth stack is split into three main parts: device handling (initializing the device, packing and sending HCI commands), HCI handling (unpacking HCI commands, HCI state machine) and L2CAP handling (unpacking L2CAP commands, L2CAP state machine). The final application interacts with the Bluetooth module using setup functions, and by passing a callback structure to the HCI state machine. The stack calls specific application functions upon device initialization, connection establishment, pincode request, received data or when it is idle, allowing the application to do its own work. Figure 8 depicts the internal organization of the stack.

The BLUEWAND Bluetooth stack was designed to support two application scenarios. In the first scenario, depicted in Figure 9, the host side contacts the BLUEWAND, which then starts transmitting sensor data. The host application makes a Bluetooth inquiry, and pages devices

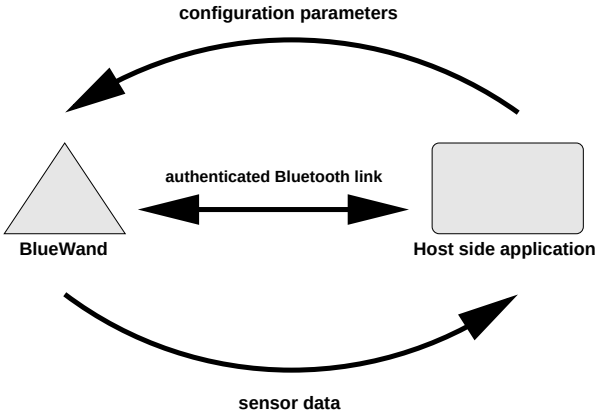


Figure 9: First scenario

matching our specifically defined BlueWand class of device. After an authenticated connection is established (the AVR ATmega103 EEPROM stores a pincode that is used to establish a common link key), the AVR ATmega103 starts sending connectionless L2CAP packets (format is shown in Figure 10) to the host side, containing the sensor data. Additionally, the host side can send configuration packets to the BLUEWAND in order to configure various parameters, such as sensor calibration, requested amount of preprocessing, etc. BLUEWAND will mostly be used in short sessions; therefore, the Bluetooth authentication algorithms, while having some critical security issues [4], is considered to be secure enough.

In the second application scenario, depicted in Figure 11, the BlueWand device periodically makes a Bluetooth inquiry, in order to find devices that offer to be controlled by the BlueWand

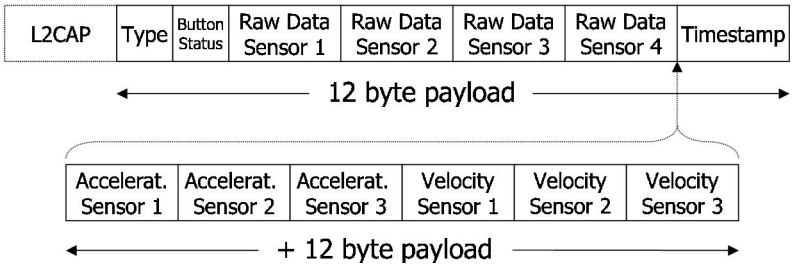


Figure 10: Data packet format

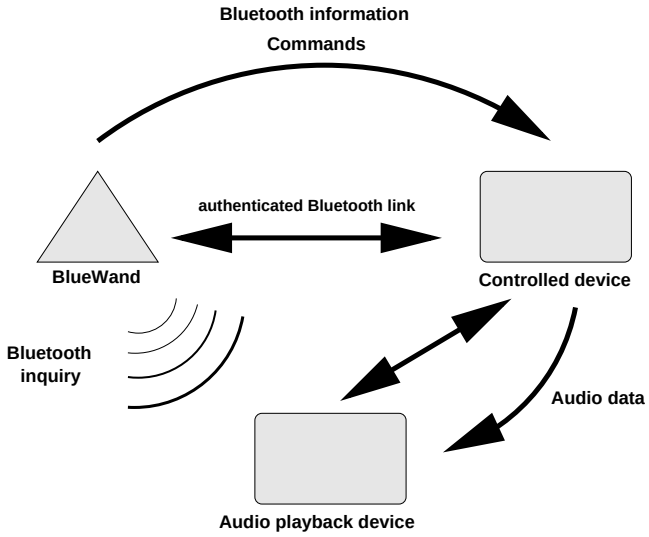


Figure 11: Second scenario

(for example, a CD player, a VCR, or a PC). If those devices are able to generate audio feedback, devices which can playback this feedback are searched. The BLUEWAND then makes an authenticated connection to the device to be controlled, sending it the information required to connect to the playback device (the Bluetooth device address, as well as the clock offset difference between its own internal clock and the remote clock). The controlled device can then efficiently connect to the playback device without having to make an inquiry. It then sends audio feedback of commands triggered by the BLUEWAND via this new connection. The same mechanism is supposed to be used to efficiently connect to a BLUEWAND that is moving, e.g. in a building. In such a scenario, the rendez-vous layer protocols for Bluetooth devices designed at the ETH Zürich [3] can be used to drastically reduce the number of required inquiries.

3.4.3 Tradeoffs and debugging

Obviously, the limited RAM- and ROM-size of the AVR ATmega103 microcontroller forced some constraints upon the design of the stack. In our first approach we used a cut-down version of the GNU libc memory allocator that was provided in the AVR libc. There upon, the Bluetooth stack was implemented using dynamic structures, for example for queuing HCI commands, or maintaining Bluetooth peer lists while doing Bluetooth inquiries. Memory usage was measured with the dmalloc library, which makes an exhaustive report of used memory, allocation frequency and wasted memory space due to fragmentation of the program it is linked with. Although the dmalloc library reported a maximal memory usage of only 800 bytes for the finished application, the Bluetooth stack ended up overwriting part of the function call stack. Thus we were forced to rewrite the stack to use static memory structures, replacing the linked lists and dynamic arrays with static arrays. Memory usage can now be controlled by defining maximum numbers for simultaneous connections and number of peers. Accurate memory measurements are given in Table 2. Our CPU time measurements showed that 95 % of the AVR application time is spent idling, allowing for massive power consumption savings by using the ATmega103 sleep capabilities (which we don't use by now), and for extended preprocessing.

While working with the Xircom CreditCard Bluetooth adapter, we had problems with serial communication to this PCMCIA card. When the throughput increased (for example when receiving sensor data), the Xircom card seemed to randomly send duplicate bytes, confusing the Bluetooth stack. Even though our Bluetooth stack was able to resynchronize with the adapter most of the time, this still was a problem, which could not be resolved by updating the

Application	Code size	Memory usage	Application parameters
AVR (stack-avr)	12600 bytes	840 bytes	(1 peer, 1 connection)
Linux/x86 (bt-mouse)	9888 bytes	2012 bytes	(15 peers, 15 connections)

Table 2: Memory requirements

firmware. This problem was also confirmed by the developers of the BlueZ stack. In order to avoid this problem we used an Ericsson module instead.

Since most of the development was done on the host side, debugging was fairly easy. On the AVR ATmega103 microcontroller, on the other hand, the only available UART was already used by the Bluetooth controller. One possible solution would have been an i^2c debug connection. But since there was no slave i^2c implementation available under Linux, we instead implemented a special gateway program that runs on a Linux PC with two serial ports. This program filters out debugging messages before the UART data is sent onward to the Bluetooth module. The AVR ATmega103 marks these messages by prepending a special ”magic key” bytestring to the message before sending them out to the UART.

3.5 The BLUEWAND Audio Player

3.5.1 Idea

Among our initial ideas for BLUEWAND was a control device for a small, portable audio player. There are several kinds of highly successful personal audio players on the market, based on either Compact Disc, MiniDisc or MP3 technologies. All of these portable players use either buttons on the device or a remote control for user input and a liquid crystal display (LCD) for information output. These personal audio players are designed to be small enough

to fit into a pocket; we believe it to be quite awkward to press buttons located directly on the device to operate them. An independent and wirelessly controlled device for command input will increase the comfort of portable audio players.

3.5.2 Concept

With the BLUEWAND, the user can operate the portable audio player using gestures to express basic commands such as: Play, Pause, Stop, Next/Previous title/album, Volume up/down.
We want the user to be able to leave the audio player in their pocket. Therefore, there is no visual feedback through the display, but through the audio channel instead. As the portable player has the hardware capabilities for audio playback anyway, both devices can be linked via Bluetooth; we simply have to store prerecorded voice samples like ”Play”, ”Stop” etc. in the player’s memory for audio feedback.

3.5.3 Gestures

The BLUEWAND is sensitive to movements in three dimensions. Additionally, it has a button the user can operate to indicate that a movement with BLUEWAND actually is a command. Otherwise, the audio player could misinterpret any random movement as a command.
We have kept gestures as simple as possible, using the six main vectors of three-dimensional space and being context sensitive. When the user gives a command through a gesture, the audio feedback of the command is played back from a prerecorded sample. If the gesture has been understood correctly, the user confirms it with the ”Yes” command. A command can optionally be canceled with a ”No” gesture or by waiting for two seconds. Gestures are given in table 3.

command	gesture
Yes (Confirmation)	quick forward movement
No (Escape)	quick backward movement
Play (from Pause or Stop mode)	quick forward movement
Stop (from Play or Pause mode)	quick backward movement
Pause (from Play mode)	quick forward movement
Increase volume	upward movement
Decrease volume	downward movement
Next title	single rightward movement
Next album	double rightward movement
Previous title	single leftward movement
Previous album	double leftward movement

Table 3: Commands of the BLUEWAND Audio Player and their gestures. Forward means a movement away from the body. A double rightward/leftward movement means to turn the BLUEWAND right/left twice within two seconds.

3.5.4 Implementation

Since we do not have a customized Bluetooth-enabled portable audio player, we modified an existing MP3 player running on a laptop computer with Linux. We used the open-source ”madplay” [7], program which is a command-line operated MP3 player, as a basis for our Bluetooth audio player. It was enhanced to receive sensor data packets from our own Bluetooth stack. Gestures are recognized by comparing the velocity peaks sent by the BLUEWAND with threshold values.

To be able to play MP3 audio files and voice command files at the same time, we use the Analog Realtime Synthesizer ”aRts” [8] from the KDE project. The artsd sound server handles playing multiple sound files simultaneously on the sound hardware.

3.5.5 Future Applications

We envision the use of BLUEWAND for other mobile devices that have restricted room for buttons and a display. Mobile phones are often operated through menus with a two-axis con-

trol. We can easily define four gestures according to this menu control. A visual feedback isn't needed when the user has a headphone connected via Bluetooth or cable to the phone. Menus can be read to the user by the phone. Even new data like telephone numbers can be entered either through gesture recognition (using BLUEWAND) or voice recognition (using the microphone). Even more, with gesture recognition and audio feedback, blind people get a simple-to-operate means for many advanced appliances.

Imagine a world were clumsy mobile phones, audio players and personal digital assistants can be kept inside your jacket or briefcase. All you need to communicate with them is a light-weight wireless headset and a BLUEWAND in your hands!

3.6 Linux Integration for BLUEWAND

So far, we have described the BLUEWAND and a single demo application. In the following, we present our approach to integrate the BLUEWAND with the Linux input system. Thereby, all Linux applications can be controlled by the BLUEWAND without being adapted.

3.6.1 Overview

Although there are already several Bluetooth stacks available for Linux, up to now no system supports a flexible management of resources necessary to handle a changing diversity of Bluetooth devices, including BLUEWAND. The Linux input core (a part of the Linux console project [5]) supports a device and driver management for keyboards, joysticks and mice. As a first step towards the integration of BLUEWAND, we started with a modular, component based device manager. Using this methodology, it was possible to encapsulate device drivers, communication and management layers into different orthogonal components and services.

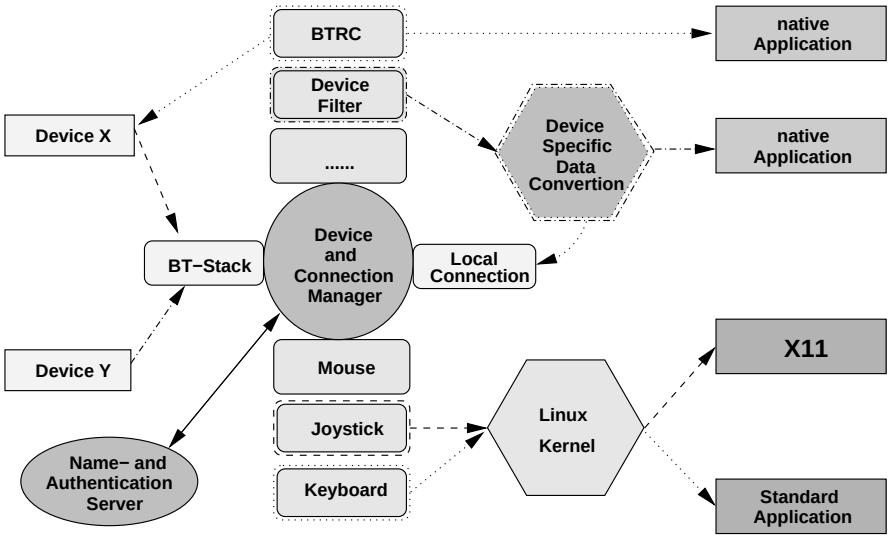


Figure 12: Possible Connection Scenario for a BLUEWAND enabled system

3.6.2 Implementation

The management system consists of a set of simple services to locate and identify devices and interfaces and connect these. The overall system is composed of three base services, a *Connection Manager*, a *Name Server* and an *Authentication Server*. These services implement a simple resource and connection management for mobile devices, their attachable drivers and specifies an easy connection interface for higher-level protocols, like BTRC [6]. Figure 12 gives a brief overview of a possible system. The service interfaces are specified in the CORBA Interface Definition Language.

3.6.3 Input and Output Devices

Input and output devices represent all kinds of external entities that can send data (input device), receive data (output device) or both. The manager views these devices as proxies which represent hardware or software interfaces. Between these it builds connections and automatically translates data from one format into another via a filter.

3.6.4 Filters

The integrated filters can be used to implement further data preprocessing extending the steps explained in chapter 3.3. The design of the device manager allows stacking of multiple filters, allowing transparent data transformation to be performed at any stage. For example, data delivered by an input device can be processed by a gesture recognition filter. The thus extracted commands are then forwarded to further local or remote devices.

3.6.5 Integration of BLUEWAND

In the course of this project, several output and filter components were implemented, for instance a bus mouse driver which can be connected to the the Gnu Pointing Manager (GPM) or the X11-server and thereby enabling the use of BLUEWAND in Linux applications. Thus, it is possible to control the mouse cursor with the BLUEWAND. This enable us to use existing programs like xstroke for script recognition while writing to a virtual white board. A special kernel module, that exports the interface of the Linux input core to the Connection Manager, was implemented in order to use the BLUEWAND with other output interfaces such as joysticks and keyboards. This way it is possible to control various Linux applications (for example the X Multimedia System [XMMS], a graphical multimedia player, or X-MAME, a video game emulator) with the BLUEWAND.

4 Summary

Our design goal behind BLUEWAND was to create a working tool that would both fit into our vision of a world of specialized tools linked via Bluetooth, and, at the same time, be realistic enough to suit production for the mass market. Special care was taken to constantly validate our design throughout the various steps of the project. Thereby, we were able to reveal and resolve several obstacles that had not been obvious in the beginning (e.g. bugs in material we tried to build upon).

By clearly defining tasks and associated personal responsability, we were able accomplish our goal in the given timeframe, and even had time left to start implementing sample applications demonstrating our vision. The Linux input core extension we developed provides useful functionality even beyond the BLUEWAND project itself.

Our future work will be focused on further improving the analysis of the sensor data, potentially leading to a graffiti alphabet of gestures, and on location-based services using stationary controlled devices as beacons. Resources in the microcontroller we used are still sufficient to make both extensions conceivable. Maybe, one day, BLUEWAND will really become a universal tool in everyday life.

References

[1] Bluetooth SIG, *Specification of the Bluetooth System (Core)*, http://www.bluetooth.com/

[2] The Open Group, *The Single UNIX Specification, version 3*, http://www.unix-systems.org/version3/

[3] Frank Siegemund and Michael Rohs, *Rendezvous Layer Protocols for Bluetooth-Enabled Smart Devices*, In Proc. ARCS 2002

[4] Juha T. Vainio, *Bluetooth Security*, http://www.niksula.cs.hut.fi/ jitv/bluesec.html

[5] Linux Console Project, http://linuxconsole.sourceforge.net/input/input.html

[6] Ivan Ivanov, Manuel Odendahl, Alexander Paar, Dr. rer. nat. Fridtjof Feldbusch, *A Bluetooth Remote Control System*, Proceedings of the 1st Int. Conference on Architecture of Computing Systems, ARCS, April 2002.

[7] Rob Leslie, *MAD: MPEG Audio Decoder*, http://www.mars.org/home/rob/proj/mpeg/

[8] KDE Project, *Analog Realtime Synthesizer*, http://multimedia.kde.org/arts.php