JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Activity Tracking with ZigBee

BACHELORARBEIT
(Projektpraktikum)

zur Erlangung des akademischen Grades

Bakkalaureus der technischen Wissenschaften

im Bachelorstudium

INFORMATIK

Eingereicht von:
Haslgrübler-Huemer Michael Josef

Angefertigt am:
Insitut für Pervasive Computing

Betreuung:
Univ.-Prof. Mag. Dr. Alois Ferscha

Mitbetreuung:
Dipl.-Ing. Dr. Simon Vogl

Linz, April 2008

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, am November 5, 2008

Michael Haslgrübler

Acknowledgment

I would like to thank Jürgen Erhart for assembling the hardware prototype and to show me the nitty-gritty of hardware-oriented C. Last but not least I would like to thank Simon Vogl for being an inspiration, a spring of motivation and not to leave me high and dry.

Abstract

The activity tracker is implemented using the IEEE 802.15.4 ZigBee wireless standard compliant MeshNetics MeshBean platform and their open source MAC layer implementation, OpenMAC, which uses the nesC programming language and TinyOS. The hardware prototype assembled uses a 3-axis-accelerometer and on the software side additional filters to detected the activity trackers state which will be transmitted via ZigBee router to a base station for further processing.

Contents

List of Figures

Chapter 1

Introduction

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"
Mark Weiser[17]

1.1 Motivation

The above statement was published by Mark Weiser[17] and has become the devise of Pervasive Computing. If one counts the numbers of device that surround him and that one does not recognize as computers anymore but more or less as long-term partners, the disappearing has already started. Within this context the activity tracker could be one of the key technologies that enable other devices to interact smartly within the user digital aura, as proposed by [5], according to the activity tracker's current state. As a context-aware application the activity tracker could be the integrated into keyring pendant and carried around just as we do it with our latchkey for everyday everywhere use. Henceforth information collected by the activity tracker could be transmitted to and used by many other devices that need specific information about their user to fulfil their task ambient and properly. Imagine running to catch the bus and being disturbed by the vibration-call function of the phone in the pocket. If a user will have the activity tracker equipped, it would send the business of its user to the mobile phone, which in turn could omit the vibration-call feature of the phone or the call itself and notify the caller that you are in a hurry and a later call may be more appropriate.

1.2 Task

The task was to create an embedded sensor node that is context-aware and collects information about the user and his behaviour. It was also required that it can recognise its own state, cf. Figure 1.1, independently from a high level processing unit. The activities we wanted to collect about the user are as follows: Idle, Sitting, Standing, Walking, Stand up, Sit down. These basic states were found to be the lowest level information that needed to be collected because higher level information, like having a meeting, could be better observed from the outside. It was required to build the application on top of MeshBeans OpenMAC, which is a open source MAC Layer implementation for ZigBee, the IEEE 802.15.4 wireless standard, on top of their hardware platform using nesC, a network centric C like language, as the primary programming language.

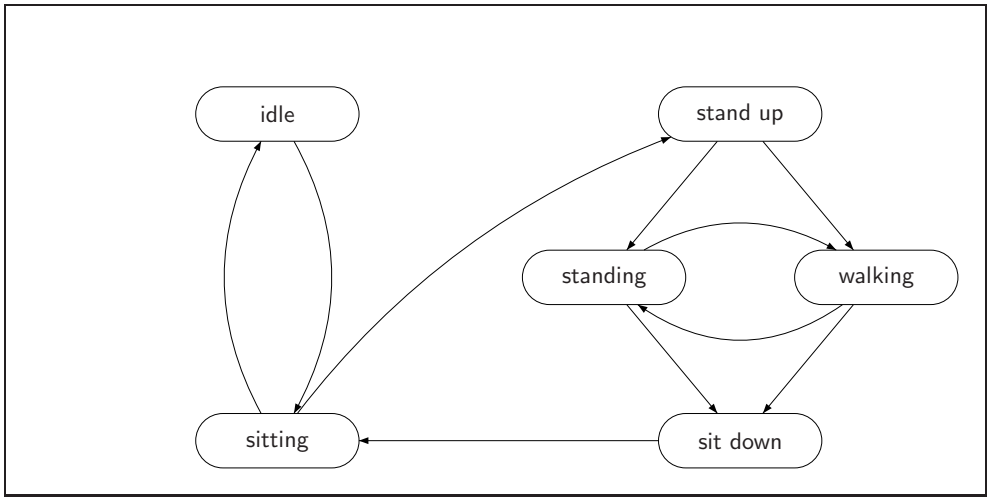


Figure 1.1: States to be recognised

1.3 Solution

The OpenMAC implementation, was extended with the necessary requirements to read data from the sensors and to send them wirelessly to a base station. There were filter libraries designed to be easily ported to other TinyOS architectures easily and suit the low-cpu requirements of an embedded system. A state machine was created to fulfil the detection of the states with the values from the sensors and filters and output the new computed state.

The remaining chapters of this thesis are structured as follows. Chapter two contains related work which was done on activity tracking over the last years. Conceptual Information is described within the system architecture, chapter three. Afterwards chapter four presents the filter information used and in general implementation details used in the project. At last future work and conclusion round off the thesis and summarize the whole work.

Chapter 2

Related Work

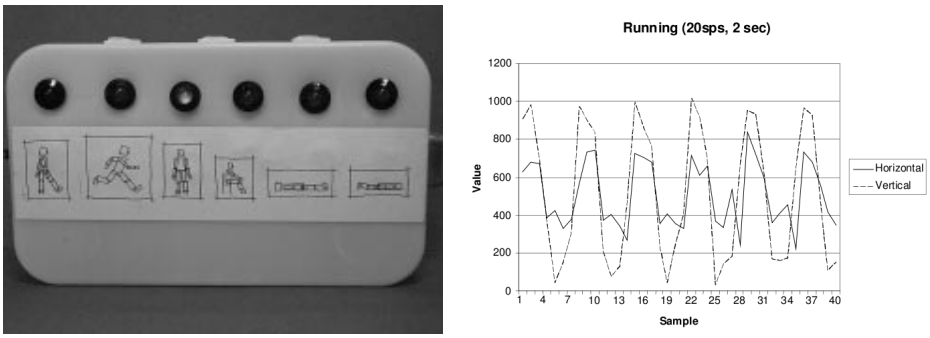
Lukowicz [11] suggests and classifies the task for recognition in different subfields, cf. Figure 2.1. Where every subfields is complex and essential for the next step. Errors made in the early steps will increase the error and the uncertainty on each further step. Its necessary to choose the right kind of sensors for different tasks with a special focus on information gain and power needed which is a main focus of embedded system designs. Within signal processing we have to choose the right sampling frequency, according to the Nyquist–Shannon sampling theorem, and may be remove the noise with a low pass filter as described in Section 4.3 and if necessary additional filters for more information. After analysing the sensors signals its necessary to choose the adequate features. Optionally pattern matching has to be applied to find reoccurring sample cycles, however this is better suited to be computed on non-battery dependent devices. At last higher level processing makes use of the collected data and uses the bits and bytes collected to discover a more abstract kind of information or knowledge.

Figure 2.1: Recognition Chain[11]

In the following sections the evolution of activity tracking will be discussed in four different project and three different approaches to solve the almost same task. However there are other approaches, e.g to use either external sensors or motion detection in videos for activity tracking, which are not covered within this paper.

2.1 Sensor Badge

The Sensor Badge [4] describes how to distinguish between a set of predefined activities using two accelerometers, one horizontal and the other one vertical. To differ between sitting, standing, lying (up and down) Farringdon et al. examined they magnitude of the acceleration (on the voltage level) and labelled the different amplitude levels to the according states. While detecting walking and running they found out that that the difference between those two states were the vertical amplitude and the horizontal difference between maximum peak and minimum peak is almost the same. The average of the values collected was similar to standing and the crossing of the average boundaries was used to obtain a frequency also indicating if the user was walking or running. The sensor badge finally outputed the detected state with labeled LEDs, cf. Figure 2.2, or via serial communication for further processing.

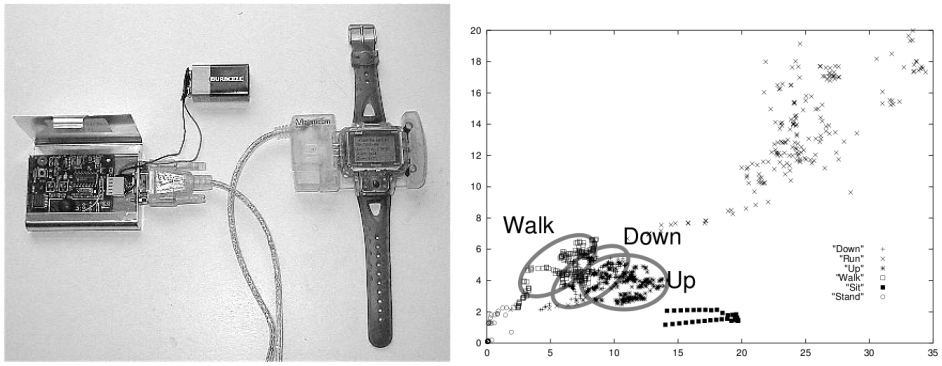


(a) *Prototype* (b) *Running Analysis*

Figure 2.2: Sensor Badge[4]

2.2 OnHand PC

Randell and Muller [16] used a different approach to differ between an other set of activities, which were walking ,running, sitting, upstairs, downstairs and standing. They used feature extraction for creating input for neural network, cf. Figure 2.3, out of a horizontal and vertical sensor and labelled the x/y values collected by the ADC with the current activity and the integrated values over the last two seconds on an ground truth of ten people.However with additional filtering the system used to recognise up to 95% right although there were some problems regarding the height of landings when walking up and downstairs and different clothing made the recognition task more complicated. The accelerometer board was event-triggered and only outputs changes of current state to the OnHand PC via serial communication.

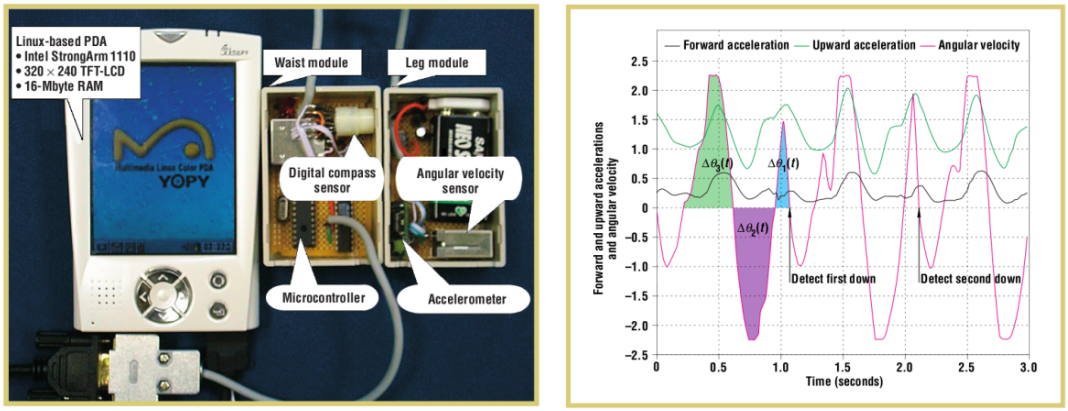


(a) *Prototype* (b) *Neural Network*

Figure 2.3: Activity Recognition with Neural Networks[16]

2.3 PDA and wearable sensors

Lee and Mase [10] used a bi-axial accelerometer, a gyroscope and a angular velocity sensor, which were attached via serial communication to a PDA, cf. Figure 2.4, in order to perceive the states sitting, walking (different speeds), upstairs, downstair and standing. Using dead reckoning, an algorithm to estimate the position using time, speed and walking heading, they did also do location recognition.



(a) *Prototype* (b) *Walking Detection*

Figure 2.4: Activity Recognition with Feature Extraction[10]

They used a feature vector with standard deviation of the accelerometer and the angular velocity sensor as well as the last three angle differences, which were obtained using integration between zero-crossings. For recognizing standing and sitting the accelerometer uses the absolute gravitational acceleration if the unit wasn't in motion. Walking detection was done by a peak detection algorithm on the upward acceleration

and the rule that the standard deviation of x/y and angular velocity. If more than two of these values were bigger then the threshold then the systems checked the number of zero-crossings on the angular velocity and if there were more than two crossings, the systems tried to find the the angular differences and successfully detect a gait cycle of human walking.

2.4 PowerSaver, Pocket-Worn Activity Tracker

Ferscha et al. [6] created a pocket-worn activity tracker, called the PowerSaver, which detects sitting,standing and walking with an accuracy of $80-90\%$ when worn on the hip. The PowerSaver is an ambient device to enable energy efficient computing, hence the name. It interacts with "Control Units" which are connected to any kind of electronic device and set them to a power saving state or shut them down completely. This interaction is done wireless and requires no interaction from the user. The prototype setup developed so far behaves smartly and adjust the light to the PowerSaver's current context, e.g. fading light when walking away.

2.5 Comparison

Four projects have been introduced to detect user behaviour, however comparison will focus on the first 3 projects since the limited information regarding the PowerSaver. The first two projects both have the disadvantage of serial wired communication for transmission of users state to other devices, the third one, on the other hand, can use the Bluetooth or Infrared wired communication of the PDA. With exception of the first and last project all projects use a lot of computational power and implicit battery power. The activity tracker resolves these issues it uses the low power wireless communication which comes through ZigBee to provide information for context-aware applications and multiple devices. Also the computational requirements of the activity tracker are better than those of the last two projects. Finally the small design will encourage the typical user to use it everyday. These advantages of the activity tracker fit the requirements and specification of a pervasive system better than the first three introduced related projects.

Chapter 3

System Architecture

The system architecture will be explained within this chapter for better understanding of the bigger picture. Within the first section the relations between the technologies and the hardware used will be explained. Followed by an introduction to the ZigBee wireless standard and platform which is the key technology and upcoming de facto standard used for wireless sensor networks. Afterwards a brief tutorial to concepts of nesC and TinyOS will be given. Finally the choosing of the Meshnetics MeshBean development platform and their OpenMAC suite will be reasoned.

3.1 The Architecture

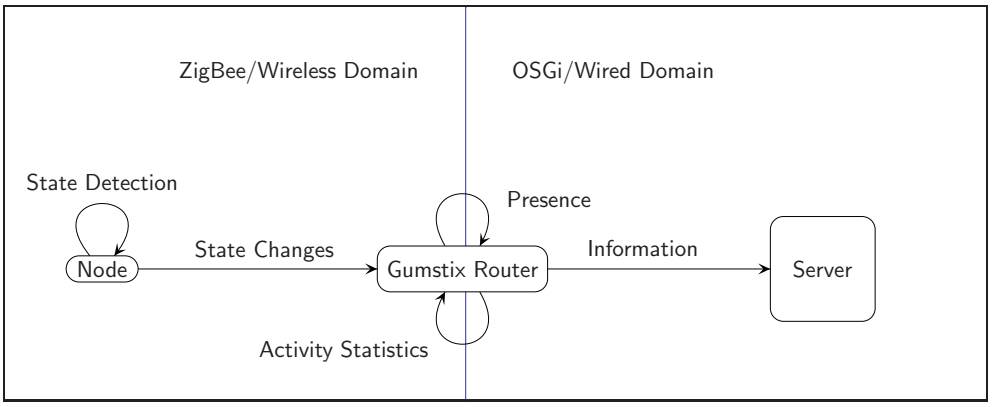


Figure 3.1: Architecture Overview

The architecture consists of the three parts so far. The sensor node, cf. Figure 3.2, were used for collecting information about the environment with a 3-axis-accelerometer and a temperature sensor which it used for state detection. In the final version only

state changes are transmitted, but currently for easier development the node sends its raw data with the detected state to the MeshBean developement board which writes the incoming data to the serial interface of the PC which it is attached to. However its also possible to attach the sensor node to the PC directly and access its data. On the PC, a Java application reads the data from the serial interface, performs some sanity tests on the values which are used to avoid transmission errors. Afterwards the values are processed by the same filters like on the end device and put into the state machine. When the filtering and state detection is done the application writes all values to the console as well as to a log file which can be used for visualisation using gnuplot, a GNU/Linux plotting tool.

Figure 3.2: sensor size comparison

The target architecture is more complex. The sensor or a lot of sensors exist(s) in a smart living environment where they hop through different network zones and attach/detach to different routers so the next best router has to take care of the activity beacons the sensor nodes transmit as well as detecting statistics and presence information about the motes. This information can be routed directly or indirectly over other routers to a base station where the information enters the OSGi/Java Domain which is the key interface to a more higher level processing domain.

3.2 MeshNetics OpenMAC

It was required, cf. Section 1.2 to build the system on the MeshBean platforms[12] which is one of the first and also open IEEE 802.15.4, MAC Layer implementation with TinyOS support which is an excellent choice for building low power devices with the quality of service level only a MAC layer provides. However setting up the toolchain isn't straight forward and will be explained later, cf. Section 4.5. The development kit boards, cf. Figure 3.3, are good starting points for developing own applications and is used in our system as a ZigBee Router, Coordinator and a interface to the PC. Jürgen

Erhart assembled the prototype of the sensor node which is displayed Figure 3.3 b and c.

(a) *MeshBean Platform* (b) *node top view* (c) *node down view*

Figure 3.3: Meshnetics MeshBean DevelopmentBoard[12] and Sensor Node

3.3 ZigBee Wireless Standard

There are many wireless standards, cf. Figure 3.4, used so far which all have their special application domain. Within this wireless space the new standard ZigBee [8, 1], which is spaced on the the IEEE 802.15.4 Radio Layer Specification[9], is a unique approach for communication between low power devices with low data rate, enforced by the ZigBee Alliance which is a collaboration effort between many hardware vendors. There are two types of devices within the ZigBee standard, Reduced Function Devices (RFD) and Full Function Devices (FFD). Reduced Function Devices can be seen as battery powered low-power devices which collect information about their surrounding optionally do some processing and send them to a Full Function Device which can be used to route the information to a base station, either directly or indirectly over other FFDs. This aspects of indirect routing is used to form ZigBee networks with different kind of topology although the idea of ZigBee networking suggests mesh topology which may consist within itself also as star/tree/mesh networks. Within a mesh network many connection a redundant so if a router fails another one takes over the job and the packages are sent to the network coordinator with the fastest speed.

Since ZigBee is a network protocol its necessary to identify devices, therefore the IEEE Standard applies a 64 Bit IEEE address, which is globally unique, and a 16 bit address which makes it possible to have 65.535 nodes within a network. With that amount of devices its necessary for ZigBee routers to have power from the grid as well as more RAM than RFD devices for their routing/transmitting abilities. The ZigBee MAC

Layer with its rather small size of 104 Bytes maximum payload provides the necessary methods to speak over the shared wireless channel.

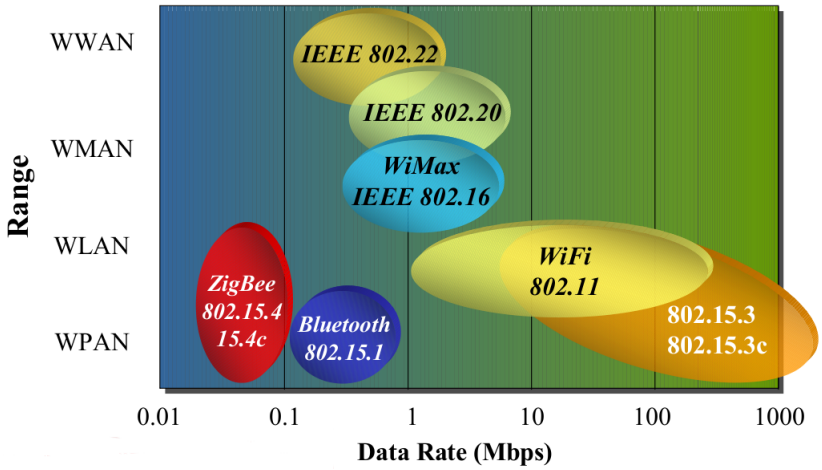


Figure 3.4: Placement of the ZigBee Standard within the IEEE Wireless Space[8]

3.4 nesC

nesC [7] is an programming language designed for **n**etwork **e**mbedded **s**ystems which extends the **C** programming language, which is required for low level access to microcontrollers, with safety, simplicity, optimisation and program-analysis. nesC discourages dynamic allocation which helps to provide more accurate information about runtime requirements and problems, such as race conditions or needed memory. The use and providing of interfaces and the component model addresses the needs for dynamic allocation.

The component model is one of the basic and sophisticated feature of nesC . Its about using and providing interface which are access point to the functionality of a component. With this design pattern its possible to provide common standardized access to different hardware specific platforms or devices.

3.5 TinyOS

TinyOS [7] is an operating system designed for the needs of network embedded systems and goes hand in hand with the nesC programming language in which it was implemented. It's build up on three concepts: Component-based architecture, which

was already explained above; Tasks and event-based concurrency; Split-Phase Operation.

Concurrency is implemented via tasks and event. Tasks are a non time critical computation mechanism which is used to defer some code execution when the operating systems thinks its appropriate. Events are highly preemptive and are commonly used to signal the end of a task which is used in split-phase operation.

Split-Phase operation, splitting up the request and the execution of a function, is another core feature of TinyOS which is necessary because of the non-preemptive way of tasks. Its necessary to introduce non-blocking operations which have to include at least two parts to allow split-phase operation within an interface. The request of such a operation is classified via the command keyword and for completion an event is signaled.

Chapter 4

Activity Tracker

The software part of the activity tracker was built around the MeshBean development platform and OpenMAC, cf. Section 3.2. In this chapter the used filter will be explained as well as the state machine which was build to map the features computed by the filter bank to an appropriate and correct state. The concepts of nesC will be shown on the example of a filter implementation and the integration of the presented parts into the activity tracker system. Finally the recommended setup of the tool-chain will be shown.

Prerequisites

For the first development stages MeshBean development platform and Debian GNU/Linux on the PC counterpart were used. For setting up the serial communication the program cKermit was used to negotiate the transfer parameters between the PC and one of the development boards, which functioned as the receiver and was attached via USB. Afterwards for rapid development Java was used to build and test different filter and to create the state machine. Those Java classes were then ported to nesC which extensive use of the concepts of TinyOS .

4.1 Sensing & Filtering

The 3-axis accelerometer, is aligned with its z axis on the vertical plane, is loaded by gravity, and x and y are on the horizontal plane. The accelerometer outputs values between zero and 1023 and the arithmetic mean, 512, is the standard value and indicates that zero pressure is on the axis. When the sensor node is standing on the side, cf. Figure 4.1, z and x axis switch to the horizontal plane and y gets loaded by the gravity

force. These changes can be analysed to detected the alignment of the sensor in order to separate standing from sitting.

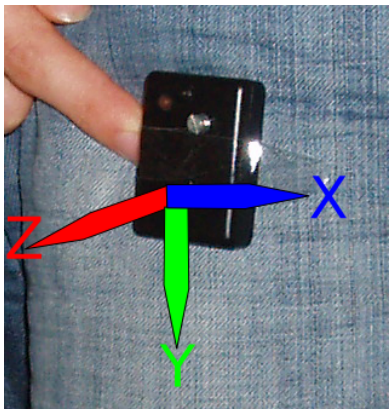


Figure 4.1: sensor axis alignment when standing

After analysing the raw signal of the sensor which samples every 80 ms, a human can match what the user was doing however if he observed what the user was doing. Learning a computer this recognition task proves more complicated. Figure 4.2 shows the sensor in different modes as follows: idle, sitting, standup, standing, walking, standing, sitdown, sitting and idle before a low pass filter which is explained in 4.1.1. For better detection algorithm two kinds of filters were implemented which were proven to be quite useful in the recognition process, a maximal filter (cf. Section 4.1.2) and a slope filter (cf. Section 4.1.3).

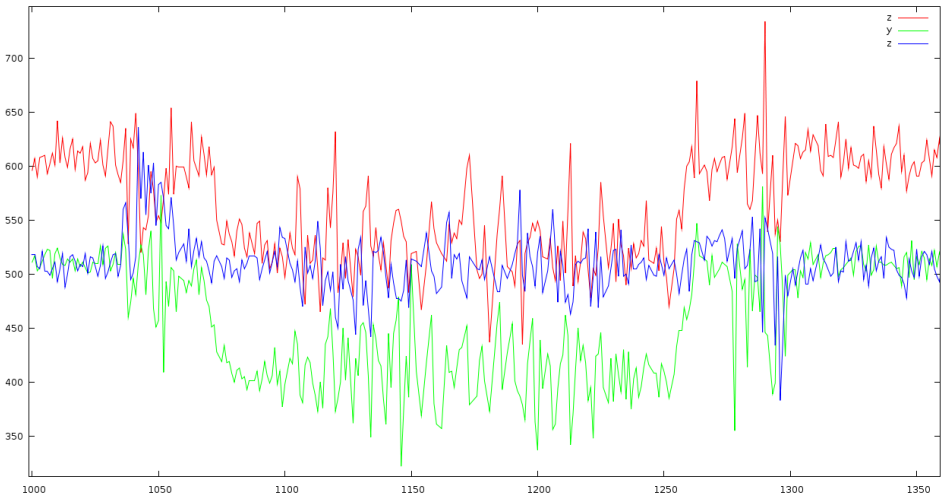


Figure 4.2: raw sensor signal when doing the testcycle[1]

[1]testcycle states: idle, sitting, standup, standing, walking, standing, sitdown, sitting and idle

4.1.1 Low Pass Filtering

Dorr [3] implemented a simple digital low-pass filter which works like an analog RC filter and suits the limitations of an embedded system. Its needed to remove the noise from the raw signal, cf. Figure 4.2, . In Equation (4.1) the mathematical relation between the incoming value $v(n)$ and the scaled output value $low(n)$ is shown and Listing 4.1 displays the equivalent C code.

$$low(n) = (1 - 2^{-k}) * low(n-1) + v(n) \tag{4.1}$$

Since the timer of the sensor fires every $80[ms]$ the sample frequency is $12,5[Hz]$ therefore it takes 1,28 samples to change the output of the low pass filter from zero to one. This delay was found out to be sufficient to remove the noise from the raw sensor value.

```
1 register = register - (register >> k) + v;
2 low  = register >> k;
```

Listing 4.1: C-Code for Low Pass Filter

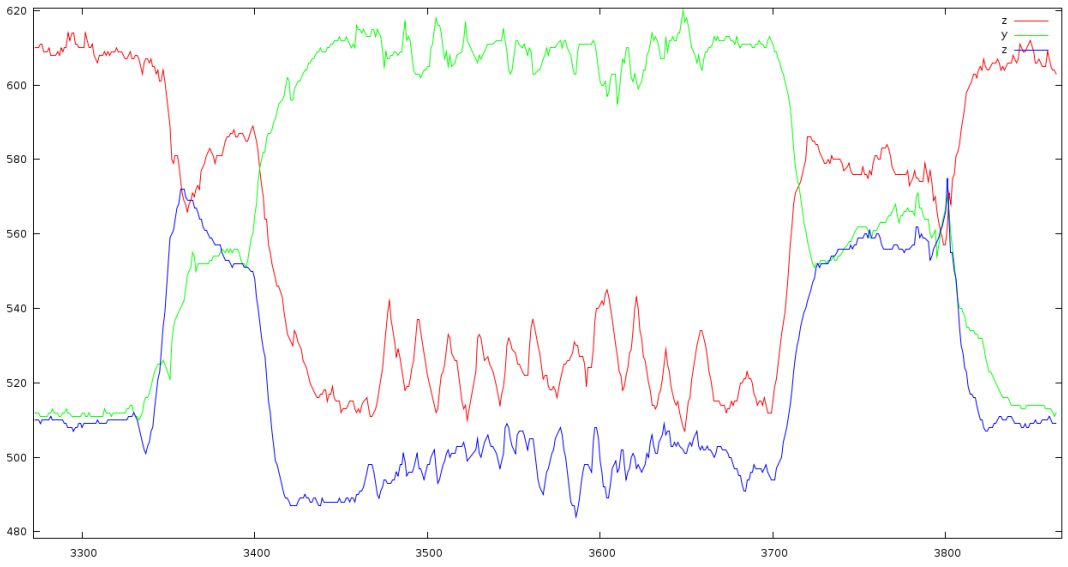


Figure 4.3: signal after applying the low-pass to the testcycle[1]

In the following parts the low pass filtered value will be named the raw value for simplicity.

[1] testcycle states: idle, sitting, standup, standing, walking, standing, sitdown, sitting and idle

4.1.2 Maximal Filter

Since the 3-axis-accelerometer measures values between zero and 1023 value the arithmetic mean, 512, is used to calculate the value of the deviation. These values are even more separated by multiplication with 30. This was found to be necessary because otherwise it would have not been possible to separate pocketing of the sensor, vertical and horizontal alignment that easily.

$$max(n) = \max_{|v(x)-512|*30} \mid n - win \leq x \leq n \tag{4.2}$$

In Figure 4.4 the raw and the max signal are displayed. Significantly only one max-filtered signal is highest which is the axis which is heavily load with gravity. Hence recognition in which position the sensor is possible, so either horizontal or vertical aligned. Also their are small staircases which could be used for estimated step calculations while walking.

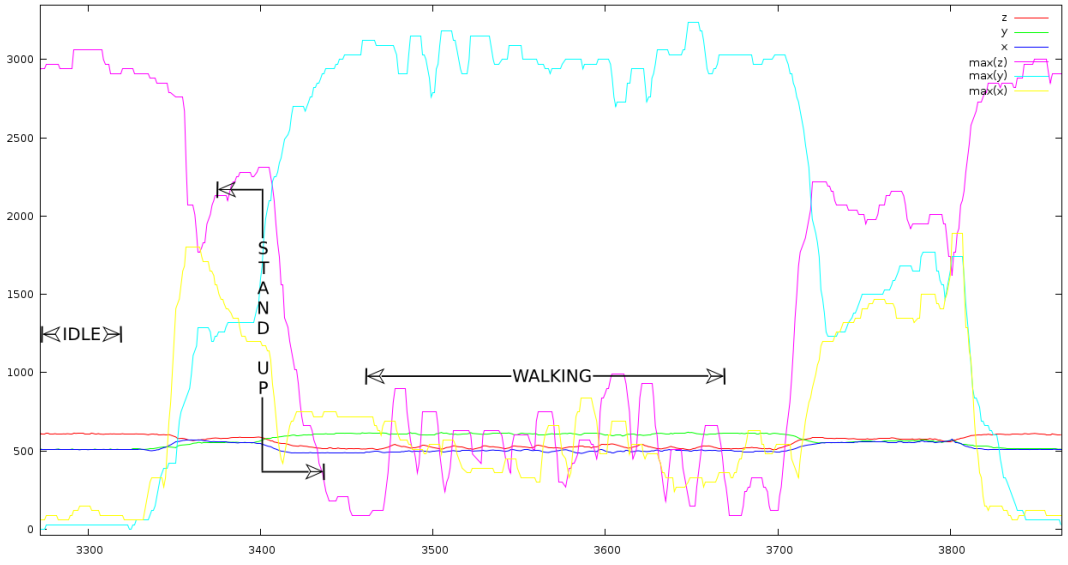


Figure 4.4: signal after applying max-filter to the testcycle[1]

The input values of the maximal filter are also used to calculate the difference between the oldest and the newest incoming value. This is mainly used for detection of stand up and sit down where the fast changes of the gravity load create a high amplitude.

[1] testcycle states: idle, sitting, standup, standing, walking, standing, sitdown, sitting and idle

4.1.3 Slope Filter

A slope filter is used to calculated the gradient over a specified window size. Figure 4.5 shows the applied slope filter, while the amplitude noise while idling is relative small compared to stand up, walking and sit down, cf. Figure 4.5 annotations,. The slope filter is mainly used to separate standing from walking. The formula used for slope filtering would be quite CPU intensive however only sums with dependency on the incoming value have to be computed every time.

$$slope(n) = \frac{win * \sum_{x=s}^{n}(x * v(x)) - \sum_{x=s}^{n} x * \sum_{x=s}^{n} v(x)}{win * \sum_{x=s}^{n} x^2 - (\sum_{i=s}^{n} x)^2} * 100 \quad | \quad s = n - win \tag{4.3}$$

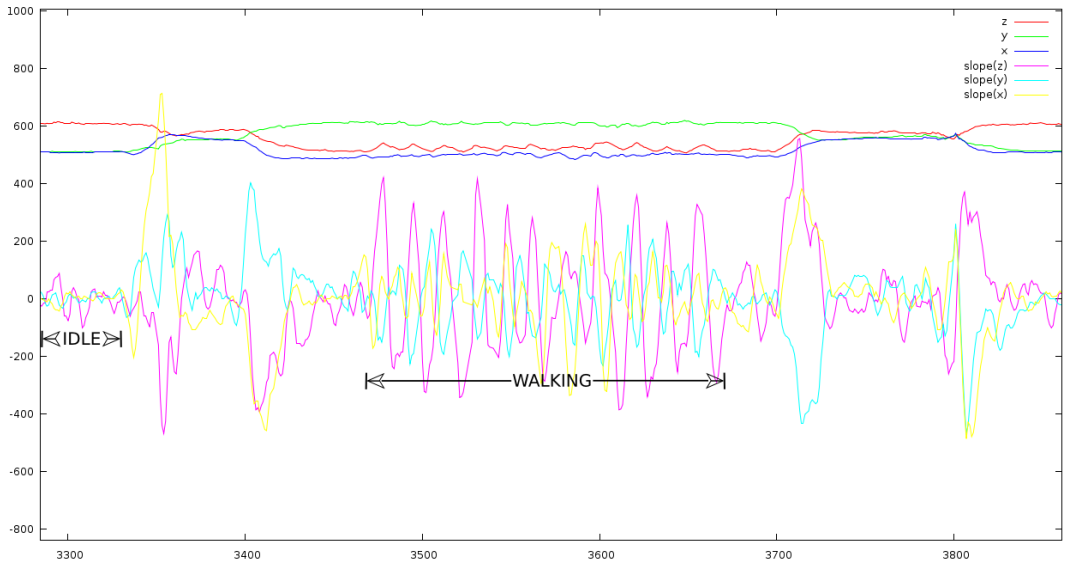


Figure 4.5: signal after applying slope-filter to the testcycle[1]

4.2 The state machine

A transducer finite-state machine, cf. Figure 4.6, was created to fulfil the requirements of activity detection. Detection starts with *try idle* and if the condition is fulfilled it changes the output state to idle, as explained in paragraph validations transition , if that is not the case it will try the next possible state, cf. indicator transitions on the next incoming values, e.g. in stand up either sitting or standing. This applies to all

[1] testcycle states: idle, sitting, standup, standing, walking, standing, sitdown, sitting and idle

states so basically it would be possible that the detected activities never change however tests have shown that this is not the case in any environmental setup.

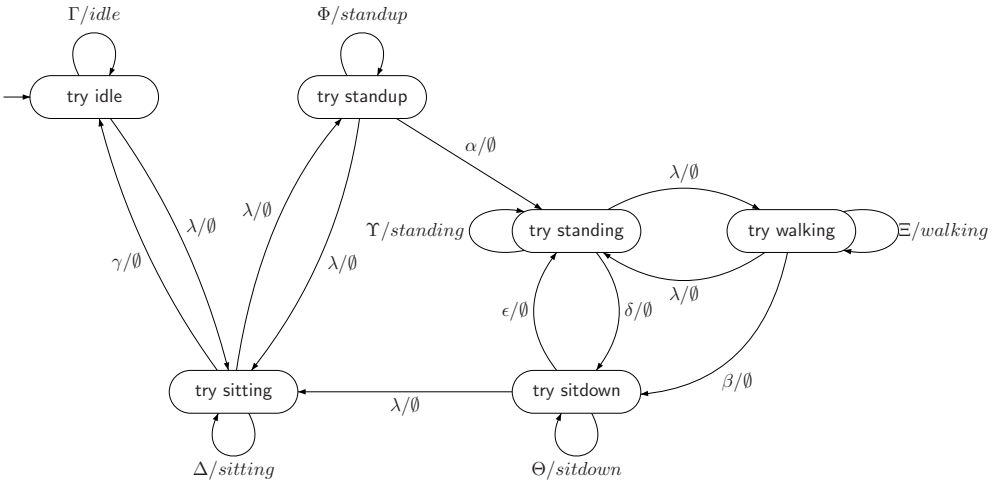


Figure 4.6: transducer finite-state machine for the activity tracker

For simplicity following nomenclature will be used $axis_{filter}$ where $axis$ is the $x/y/z$-axis used and $filter$ the $raw/max/slope$-filter applied. Also the acceleration of gravity g will be redefined in equation 4.4 to the value inflicted, which was measured, on the max filter.

$$g = 9.81m/s^2 \approx 3100_{max} \tag{4.4}$$

Validations Transition

The following transitions, cf. Figure 4.6, are validation transitions which means the currently "try" state will be evaluated by these conditions and if fulfilled trigger the transition and the according output. If this is not the case the indicator transition will be tried.

- $\Gamma/idle$ if the raw values are within a deviation of 20 around 515 for x_{raw}, y_{raw} and 615 for the gravity burdened z_{raw} axis.
- $\Delta/sitting$ unlike idle in sitting the sensor is not perfectly parallel to the ground so the max value of the gravity axis, z_{max}, has to be less than normal but more than $\frac{1}{2} \cdot g$ and either x_{max} and y_{max} above $\frac{1}{5} \cdot g$ since the load of the gravity is distributed on one of these axis.
- $\Xi/walking$ if x_{slope},y_{slope} or z_{slope} is above 150 or below -150, which are the observed lower boundaries for different persons walking behaviour, and x_{max} or y_{max} are loaded with more than $\frac{2}{3} \cdot g$, because on of those axis is loaded with

gravity force, and the difference, between the first and last value, on the x_{max} axis is lower than $\frac{1}{4} \cdot g$.

- $\Upsilon/standing$ is similar to walking x_{max} or y_{max} have to be loaded with $\frac{2}{3} \cdot g$ but x_{slope},y_{slope} and z_{slope} should be in between of the lower walking behaviour boundaries.
- $\Phi/standup$ if x_{max} or y_{max} have to be more than $\frac{1}{3} \cdot g$, which is the indicator of the beginning gravity force shift towards standing, the gain on z_{max} has to be more than $\frac{1}{4} \cdot g$ and the last detected state has to be either *sitting* or *standup*
- $\Theta/sitdown$ if z_{max} is loaded with more than $\frac{1}{3} \cdot g$, beginning back shift to normal gravity burden on the z axis, and z_{max} raise is more than $\frac{1}{4} \cdot g$.

Indicator Transitions

Basically the following transitions, cf. Figure 4.6, except the last one are indicator transitions which means that according to the most common fulfilled conditions of the following try state they are validated.

- $\alpha/\emptyset$ if x_{max} or y_{max} exceeds more than $\frac{2}{3} \cdot g$. Indication $\Rightarrow$ standing.
- $\beta/\emptyset$ if the gain on z_{max} is more than $\frac{1}{4} \cdot g$. Indication $\Rightarrow$ sitting down.
- $\gamma/\emptyset$ if x_{max} and y_{max} fall below $\frac{1}{6} \cdot g$. Indication $\Rightarrow$ idling.
- $\delta/\emptyset$ if z_{max} is greater than $\frac{2}{3} \cdot g$ and the rise on z_{max} is bigger than $\frac{1}{4} \cdot g$ Indication $\Rightarrow$ sitdown.
- $\epsilon/\emptyset$ if x_{max} or y_{max} are greater than $\frac{2}{3} \cdot g$. Indication $\Rightarrow$ standing.
- $\lambda/\emptyset$ if any other transition is possible

Example

In Figure 4.7 a labelled example shows the most significant curves and the state of the sensor and the state calculated by the computer. The first down on y_{slope} is the pocketing of the sensor followed by the stand up (annotated sitdown) which is detected successfully by both machines. However when switching from stand up to standing the pc makes an error which is possibly caused by an error during transmission. These errors occur also sometimes while walking but they do not change the overall trend of the activity detection. During the switching from standing to sit down (around $y = 39650$, annotated sitdown) a error occurs which is caused by the rising edge which suggests the user is walking again however because of the rising of the z_{max}, which is

caused by rising acceleration of gravity back on z_{max}, it is then detected that the user is sitting down again.

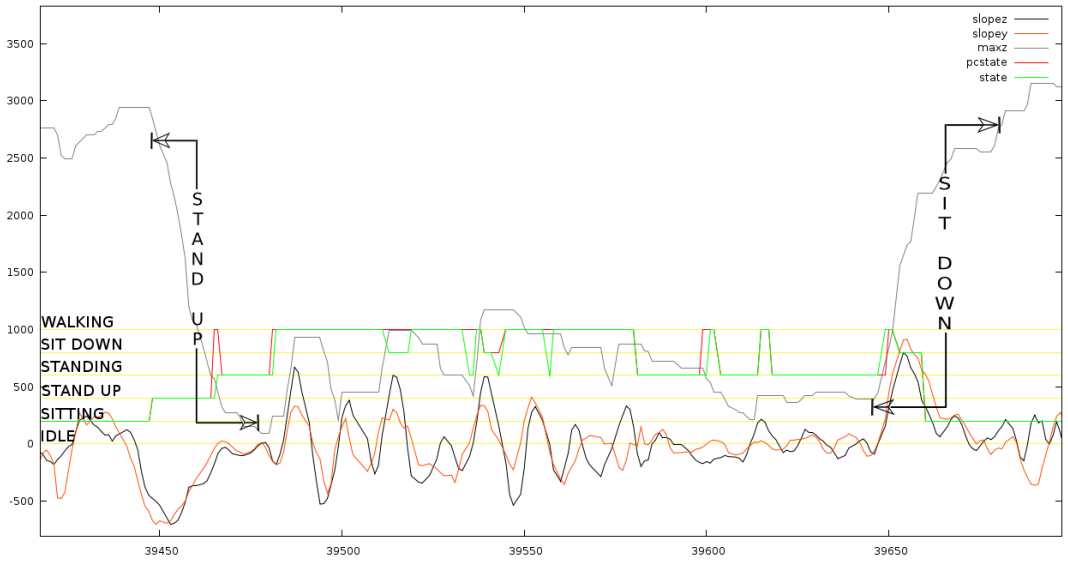


Figure 4.7: activity labelled example with selected curves for easier viewing

4.3 System implementation & integration

Every time the timer fires within the sensor node also called transmitter the ADC conversion of all axis, temperature and battery level starts, cf. Figure 4.8, . Additionally, the callback function of the ADC also applies the low pass filter. Afterwards the filter-bank will be called with new computed values which is integrated in the TinyOS part of the OpenMAC software stack, cf. Listing 4.2. Subsequently the state-machine is called with the raw and filtered values. When the new state is computed the out(put) event is signaled and the state updated. Finally sendPacket will pack and ship the activity beacon, cf. Figure 4.9, to the transmitter. Packaging is needed because of the 8 Bit fragmentation of the OpenMAC stack packages.

```
1 openmac-1.4.1.1/src/Stack/development/tos/system
2   interfaces/MaxFilter.nc
3   interfaces/SlopeFilter.nc
4   interfaces/StateMachine.nc
5   StateMachineC.nc
6   SlopeFilterC.nc
7   MaxFilterC.nc
```

Listing 4.2: filter-bank integrated within the OpenMAC software stack

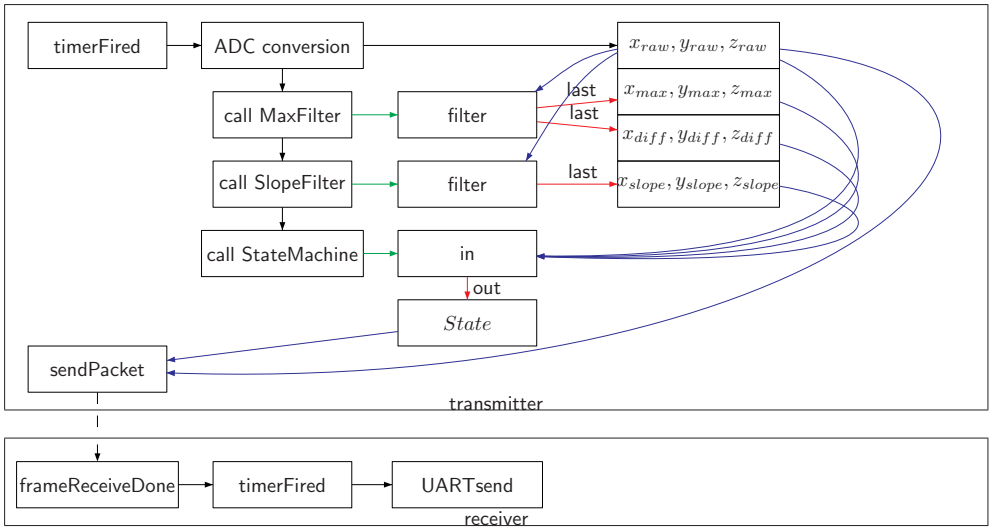


Figure 4.8: Program flow, red = signaled events, blue = used variables, green = called commands

Receiver

When a package is received on the other side the function frameReceivedDone is executed the package is unpacked and within timerFired the line for serial communication output is built and outputted via the function UARTSend. The received signal strength indicator, RSSI, which is a parameter of the frameReceivedDone is also appended to the UART output and could be used for quality of service when routing as well as sensor location detection using the (WLAN-)fingerprinting technique. The serial output contains the values as follows seperated via semicolon. *count*, z_{raw}, y_{raw}, x_{raw}, *temperature*, *battery*, *state* and RSSI value. The Java Program reads the input and performs filtering and outputs *battery*, *count*, z_{raw}, y_{raw} ,x_{raw} ,*temperature*, *battery*, z_{slope}, y_{slope}, x_{slope}, z_{max}, y_{max}, x_{max}, z_{diff}, y_{diff}, x_{diff}, $PCstate$ and *state* which can then be used to be visualized with various gnuplot commands, cf. Appendix B.

Activity package

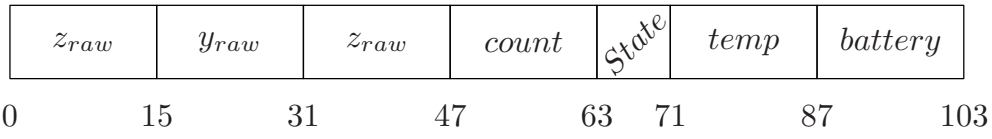


Figure 4.9: Activity Package/Beacon

The package, cf. Figure 4.9, consists of 102 Bits at the moment and can be extended with a more information if necessary however it should not exceed the specification of 102 Bytes. The raw sensor values of all axes, the temperature, battery level, the computed state and a counter are transported from the sensor to the base station. The package counter indicates when the frame was sent and will be used for routing purposes to omit broadcast cycles when being in a multi-router environment.

4.4 Hands-on experience with nesC

As nesC extends the normal C code in the following it will be explained how to use some of the advanced feature of nesC on the example of the slope filter implementation.

```
1 interface SlopeFilter {
2  command result_t init();
3  command result_t filter(uint16_t value);
4  async event result_t last(int16_t slope);
5 }
```

Listing 4.3: nesC interface for SlopeFilter

In Listing 4.3 the interface to the slope filter is shown. Its also the default interface for any filter and needed to interact with the filter implementation, cf. Listing 4.4 . The implementation provides us with a array implementation of slope interfaces which can be referenced with a given id. Within the main application configuration file, cf. Listing 4.5 the ids are matched to symbolic names in order to be used them in the application. Hence its not necessary to use the id on the function calls but the symbolic name does that accordingly. `call SlopeX.filter(filter.rawX);` actually would be matched to `SlopeFilterC.filter[1](filter.rawX);` internally. The implementation uses mainly arrays e.g. for the incoming values which have to be stored according to the id, SLOPEAXIS, and the window size.

```
module SlopeFilterC{
 provides interface SlopeFilter[uint8_t id];
}
implementation{
 enum {
  SLOPEWINSIZE = 7,
  SLOPEAXIS = 3,
 };
 ...
 uint16_t ys[SLOPEAXIS][SLOPEWINSIZE];
 ...
 command result_t SlopeFilter.init[uint8_t id](){
  ...
  return SUCCESS;
 }
 command result_t SlopeFilter.filter[uint8_t id](uint16_t value){
  ...
  return SUCCESS;
 }
 default async event result_t SlopeFilter.last[uint8_t id](int16_t slope){
  return FAIL;
 }
}
```

Listing 4.4: nesC Implementation for SlopeFilter

```
configuration transmitterC{
}
implementation{
 components transmitterM, ... ,SlopeFilterC;
 ...
 transmitterM.SlopeZ -> SlopeFilterC.SlopeFilter[0];
 transmitterM.SlopeX -> SlopeFilterC.SlopeFilter[1];
 transmitterM.SlopeY -> SlopeFilterC.SlopeFilter[2];
 ...
}
```

Listing 4.5: nesC Binding for SlopeFilter

In Listing 4.4 , the default handler for event functions which will handle unused events, failing is the standard way to be used. The main application implements a non-default handler, which will be called when the filter function is done, that will return success. This feature is necessary for programmer to recognize errors in his code.

```
async event result_t SlopeX.last(int16_t slope){
 atomic{
  filter.slopeX=slope;
 }
 return SUCCESS;
}
```

4.5 Environment Setup

Setting up the environment for nesC and OpenMAC can be quite problematic and will be explained briefly. First getting the CDK-AVR[2] kit which provides a working cross-development platform compiler which has to be used to (re-)compile Meshnetics OpenMAC stack as well as gcc-3.4, is mandatory. Then it is necessary to install the nesC compiler with applied patches from Meshnetics which is included in a the file

nesc−1.1−1.2.15_lux.i386.rpm within the OpenMAC suite. It should be extracted to a directory /path/to/ without the usr folder. It's not recommended to install it system-wide because it will conflict with the normal TinyOS suite.

```
1 export CC=gcc-3.4
2 export PATH='/path/to/local/:/opt/cdk4avr/bin/
3   :/usr/bin/:/bin:/usr/local/bin'
```

Listing 4.6: path variables to be set

Since our sensorboard is able to use four ADC converters we need to define the gpio pin name for the 4th ADC, cf. Listing 4.6

```
1 /path/to/openmac-1.4.1.1/src/HAL/HAL_R5/base/include/gpio.h
2 #define GPIO_ADC_INPUT_4 26    //ADC_INPUT_4 pin name
```

Listing 4.7: enable 4th ADC on sensor board

After setup of the environmental variables, cf. Listing 4.6 , and a small adjustment to the Makefile, cf. Listing 4.8 , you should have a fully functional development environment.

```
1 WSN_PATH = /path/to/openmac-1.4.1.1/src
```

Listing 4.8: Makefile Path Changement

Chapter 5

Future Work & Conclusion

What has to be done is to get routing fully functional and build the suited environment around the activity tracker to see it working while changing routers according to the signal strength and to create the wrapper around the router to send the messages as proposed in Section 3.1. It's also necessary to create an application in which the contextual information of the activity tracker can be used to fulfil more higher level goals then tracking. In the following a few example applications are described.

5.1 Example Scenarios

One of the most interesting features is the low power collective communication within a sensor network. An example application would be a user with the proposed activity tracker on his keyring and a ZigBee interface on the mobile phone. So if one of the devices leaves the home sensor network it could give and alarm to inform the user that he left his mobile phone at home or turn off all the lights, left on, in the house as proposed by Ferscha et al. [6].

Another scenario would be to extend the activity tracker with gesture detection. Using a button to switch between these two modes a user could use the device to operate the nearest enhanced ZigBee device, e.g. a HiFi-station or Media Center PC.

A third application could also be tracking of (elderly) patients with walking problems which equip the sensor and a cell phone with a ZigBee module so if the sensor detects that its wearer is falling down. It could tell the mobile phone to send a emergency call. This setup could also be used in hospital which could integrate a wireless sensor network for monitoring their patients.

5.2 Conlusion

The task was to create software for ZigBee powered device to fulfil basic recognition for activity tracking. Four projects were presented which feature state of the art technologies and hardware to fulfil a similar task. My project was to use the MeshBean Platform and the OpenMAC software stack to complete this task, additional it was required to create software using the nesC programming language and TinyOS. After getting the ADC working and the transmission of packets working to speed up development Java was used and to develop the necessary tools which were then reimplemented in nesC . Three filters, namely a low pass, slope and a maximal filter and a transducer finite-state machine that can conclude using the raw and filtered values the state in which the activity tracker is in, were created and tested to function properly and suit the requirements of an embedded system. Finally three sample applications were proposed that would suit the activity tracker.

Bibliography

[1] ZigBee Alliance. Zigbee overview. *ZigBee Alliance Tutorial*. URL `http://www.zigbee.org/imwp/idms/popups/pop_download.asp?ContentID=12394`. Online, viewed March 2008.

[2] CDK4AVR-Team. Cdk4avr – avr cross development kit. URL `http://cdk4avr.sourceforge.net/`. Online, viewed March 2008.

[3] Barry L. Dorr. A simple software lowpass filter suits embedded-system applications, 5 2006. URL `http://www1.edn.com/article/CA6335310.html`. Online, viewed March 2008.

[4] Jonny Farringdon, Andrew J. Moore, Nancy Tilbury, James Church, and Pieter D. Biemond. Wearable sensor badge and sensor jacket for context awareness. In *The proceedings of The Third International Symposium on wearable Computers*, pages 107–113, Los Alamitos, CA, USA, 1999. IEEE Computer Society. doi: http://doi.ieeecomputersociety.org/10.1109/ISWC.1999.806681.

[5] Alois Ferscha, Manfred Hechinger, Rene Mayrhofer, Marcos dos Santos Rocha, Marquart Franz, and R. Oberhauser. Digital aura. In *Advances in Pervasive Computing. A Collection of Contributions Presented at the 2nd International Conference on Pervasive Computing (Pervasive 2004)*, volume 176, pages 405–410, Vienna, Austria, April 2004. Austrian Computer Society (OCG). ISBN 3-85403-176-9. URL `http://www.ocg.at/publikationen/books/volumes/sr176.html`.

[6] Alois Ferscha, Bernadette Emsenhuber, Stefan Gusenbauer, and Bernhard Wally. Powersaver: Pocket-worn activity tracker for energy management. In J.E. Badram et.al. Eds, editor, *Adjunct Proceedings of the 9th International Conference on Ubiquitous Computing (UBICOMP 2007)*, pages 321–324, Innsbruck, Austria, September 2007. ISBN 978-3-00-022600-7. URL `http://www.ubicomp2007.org/program/video/`.

[7] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. *SIGPLAN Notes*, 38(5):1–11, 2003. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/780822.781133.

[8] Bob Heile. Wireless sensors and control networks: Enabling new opportunities with zigbee. *Embedded Systems Show 2006*, October 2006. URL `http://`

www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=9561. Online, viewed March 2008.

[9] IEEE. Ieee 802.15.4-2006 specifiction. URL http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf. Online, viewed March 2008.

[10] Seon-Woo Lee and Kenji Mase. Activity and location recognition using wearable sensors. *IEEE Pervasive Computing*, 1(03):24–32, 2002. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2002.1037719.

[11] Paul Lukowicz. Lecture notes pervasive system development ws2007/08. URL http://www.pervasive.jku.at/Teaching/lvaInfo.php?key=96. Online, viewed March 2008.

[12] MeshNetics. Openmac. URL http://www.meshnetics.com/opensource/mac/. Online, viewed March 2008.

[13] Wearable Computing Lab of ETH Zuerich and Embedded Systems Lab of University Passau. Context recognition network (crn) toolbox, 2007. URL http://crnt.sourceforge.net. Online, viewed March 2008.

[14] Sajdl Ondrej, Bradac Zdenek, Fiedler Petr, and Hyncica Ondrej. Zigbee technology and device design. *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, 0:129, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/ICNICONSMCL.2006.233.

[15] C. Randell, C. Djiallis, and H. Muller. Personal position measurement using dead reckoning. In *In Proceedings of The Seventh International Symposium on Wearable Computers*, October 2003. URL http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2000009.

[16] Cliff Randell and Henk Muller. Context awareness by analysing accelerometer data. In Blair MacIntyre and Bob Iannucci, editors, *The Fourth International Symposium on Wearable Computers*, pages 175–176. IEEE Computer Society, 2000. URL http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=1000537.

[17] Mark Weiser. The computer for the 21st century. *Scientific American*, 265 No. 9: 66–75, 1991.

Appendix A

SlopeFilter Implementation

Full source for the SlopeFilterC.nc

```
1  module SlopeFilterC{
2   provides interface SlopeFilter[uint8_t id];
3  }
4  implementation{
5   enum {
6         SLOPEWINSIZE = 7,
7    SLOPEAXIS = 3,
8   };
9   uint32_t sumxy;
10  uint16_t  sumxx;
11  uint16_t  sumx;
12  bool once;
13  uint32_t  sumy[SLOPEAXIS];
14  uint16_t ys[SLOPEAXIS][SLOPEWINSIZE];
15  int32_t tmp;
16  int16_t divisor;
17  command result_t SlopeFilter.init[uint8_t id](){
18   atomic{
19   uint8_t i, x;
20   if(!once){
21            sumxy = 0;
22    sumx = ((SLOPEWINSIZE - 1) * SLOPEWINSIZE) / 2;
23    for (i = 0; i < SLOPEWINSIZE; i++) {
24      sumxx += i * i;
25    }
26    divisor=SLOPEWINSIZE * sumxx;
27    divisor-= sumx * sumx;
28    once=TRUE;
29   }
30
31     for (i = 0; i < SLOPEWINSIZE; i++) {
32     ys[id][i] = 0;
33   }
34   }
35   return SUCCESS;
36  }
37
38  command result_t SlopeFilter.filter[uint8_t id](uint16_t value){
39   uint8_t i;
40   sumy[id] -= ys[id][0];
41   sumy[id] += value;
42   for (i = 0; i < SLOPEWINSIZE - 1; i++) {
43    ys[id][i] = ys[id][i + 1];
44   }
45   ys[id][SLOPEWINSIZE - 1] = value;
46   atomic{
47   sumxy = 0;
48   for (i = 0; i < SLOPEWINSIZE; i++) {
49    sumxy += i * ys[id][i];
50   }
51   tmp=SLOPEWINSIZE * sumxy;
52   tmp-= sumy[id] * sumx;
```

```
53    tmp*=100;
54    signal SlopeFilter.last[id](tmp/divisor);
55    }
56    return SUCCESS;
57  }
58
59  default async event result_t SlopeFilter.last[uint8_t id](int16_t slope)
60  {return FAIL;}
61 }
```

Appendix B

Gnuplot Visualisation

Various Gnuplot Commandos for viewing the log files

```
885 39537 536 417 527 764 885 203 253 -225 720 3390 870  390 420 240 1000 1000
962 39538 548 416 524 765 962 446 339 -289 1080 3390 870 840 480 510 1000 1000
937 39539 551 416 519 766 937 589 335 -282 1170 3390 750 960 480 540 800 800
896 39540 549 417 523 764 896 585 285 -210 1170 3390 690 900 540 360 800 800
964 39541 546 414 521 765 964 442 128 -139 1170 3210 540 630 270 270 800 800
772 39543 534 417 527 764 772 160 35 -25 1170 3000 450 240 150 60 800 600
775 39545 531 408 536 764 775 -171 -96 125 1170 3120 720 150 270 270 1000 1000
771 39546 525 405 531 764 771 -442 -175 210 1170 3210 720 690 330 210 1000 1000
938 39547 522 405 529 766 938 -535 -225 217 1170 3210 720 870 330 300 1000 1000
911 39549 527 428 525 764 911 -439 10 92 1110 3210 720 660 330 60 1000 1000
```

Listing B.1: Logfile sample

```
 plot 'logger.txt' using 2:8 with lines title 'slopez',\
      'logger.txt' using 2:9 with lines title 'slopey',\
      'logger.txt' using 2:10 with lines title 'slopex',\
      'logger.txt' using 2:11 with points title 'maxz',\
      'logger.txt' using 2:12 with points title 'maxy',\
      'logger.txt' using 2:13 with points title 'maxx',\
      'logger.txt' using 2:14 with lines title 'diffz',\
      'logger.txt' using 2:15 with lines title 'diffx',\
      'logger.txt' using 2:16 with lines title 'diffy',\
      'logger.txt' using 2:17 with lines title 'pcstate',\
      'logger.txt' using 2:18 with lines title 'state'
```

Listing B.2: full filtered commando

```
plot 0 notitle  ls 1, 200 notitle ls 2, 400 notitle ls 1,\
           600 notitle ls 2, 800 notitle ls 1, 1000 notitle ls 2,\
           'logger.txt' using 2:8 with lines title 'slopez',\
           'logger.txt' using 2:9 with lines title 'slopey' ,\
           'logger.txt' using 2:11 with lines title 'maxz',\
           'logger.txt' using 2:17 with lines title 'pcstate',\
           'logger.txt' using 2:18 with lines title 'state'
```

Listing B.3: Visualisation used in example section