On-Body Activity Recognition in a Dynamic Sensor Network

Clemens Lombriser
Wearable Computing Lab
ETH Zürich
Zürich, Switzerland
lombriser@ife.ee.ethz.ch

Nagendra B. Bharatula
Wearable Computing Lab
ETH Zürich
Zürich, Switzerland
bharatula@ife.ee.ethz.ch

Daniel Roggen
Wearable Computing Lab
ETH Zürich
Zürich, Switzerland
droggen@ife.ee.ethz.ch

Gerhard Tröster
Wearable Computing Lab
ETH Zürich
Zürich, Switzerland
troester@ife.ee.ethz.ch

ABSTRACT

Recognizing user activities using body-worn, miniaturized sensor nodes enables wearable computers to act context-aware. This paper describes how online activity recognition algorithms can be run on the *SensorButton*, our miniaturized wireless sensor platform. We present how the activity recognition algorithms have been optimized to be run online on our sensor platform, and how the execution can be distributed to the wireless sensor network.
The resulting algorithm has been implemented as a custom, platform-specific executable as well as integrated into TinyOS. A comparison shows that the TinyOS executable is using about 7kB more code memory, while both implementations classify the activity in up to 18 classifications per second.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Distributed Systems; I.5 [**Computing Methodologies**]: Pattern Recognition

1. INTRODUCTION

Recognizing user's actions is one aspect of context awareness which may give future generation smart appliances and mobile devices the ability to interact with users in a more intuitive way. The activities of a person may be recognized by small, wirelessly interconnected sensor nodes worn on the person's body [24]. To be unobtrusive, these sensor nodes must integrate seamlessly into the wearer's clothes. Therefore they need to be of small dimensions and can only offer very limited computational power and memory. Previous work has shown that online activity recognition of activities is possible on such miniaturized devices as the *SensorButton* [3, 23].
The context information determined on one network node is more valuable if it can be shared to other devices on the body or in the environment, such that they can react on this information. This paper presents our efforts to bring context recognition algorithms to sensor nodes acting in a network on the body and its environment. This adds additional challenges, as the type and number of available sensors is difficult to foresee and thus, the context algorithms implemented need to be made adaptable to the environment. We developed a framework that supports the dynamic adaptation of algorithms by reconfiguration of sensor nodes, called *Titan* [19]. It allows to express context recognition algorithms using interconnected data processing tasks. *Titan* executes those task graphs in a dynamically changing sensor network by reconfiguring individual sensors to update the network algorithm execution. We have extended *Titan* with pattern classifiers and enable it to recognize user activities.
In this paper we present how TinyOS [12] is ported to the *SensorButton*, such that the Titan framework can be used on it. We then show how we have selected different parameters for recognition algorithms, which are suited for this platform. Finally we evaluate the costs of using an OS instead of a custom implementation of the activity recognition algorithms.
The paper is organized as follows: Section 2 describes the *SensorButton* and how we ported TinyOS to it. The activity recognition algorithm is developed in section 3 and its implementation analyzed in section 4. In section 5 we describe how the context information computation is distributed over the network. Finally we put our work into a larger context in section 6 and conclude it in section 7.

2. THE SENSORBUTTON

The hardware used for this project is the *SensorButton*. Its architecture is shown in figure 2 and contains 4 main parts: an analog sensory circuits, digital processing, radio communication, and a hybrid power supply. The *SensorButton* is composed of two stacked PCBs of 31 mm in diameter and 11 mm in height. This makes it small enough to be worn on the body. Figure 1 shows the *SensorButton* in its wrist-mount casing, which also includes a 150 mAh lithium-ion battery. The wrist is an ideal place for the recognition of activities using the hands [24, 20]. People are already used to wear devices similar to watches and thus are not alienated by the sensation of the sensors.

The upper board of the *SensorButton* carries 3 sensors: a 3-axis accelerometer, a MEMS microphone, and a visible light sensor. All sensors can be turned off individually if they are not needed; the microphone is filtered by a second order Butterworth filter. The upper board also provides the

DOI 10.4108/bodynets.2007.114

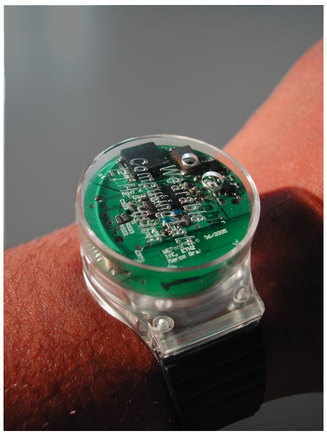


Figure 1: The *SensorButton* in its wrist-mount case

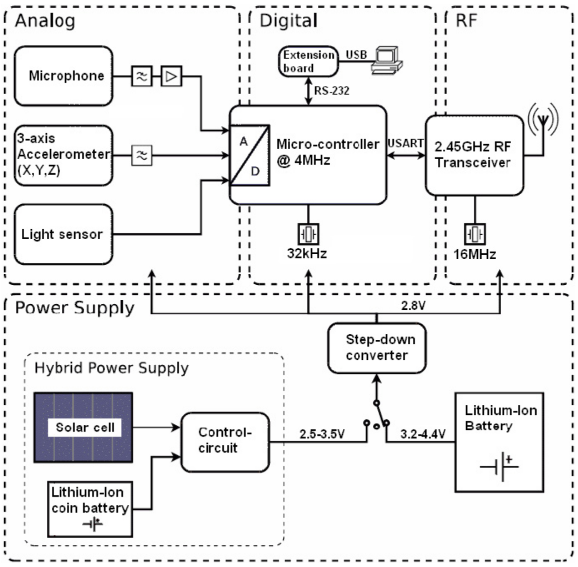


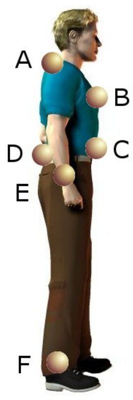
Figure 2: The *SensorButton* architecture with its 4 main parts: analog circuits, digital processing, RF communication, and the hybrid power supply

wireless communication in form of a nRF2401E transceiver working in the 2.4 GHz band. It communicates via a PCB antenna to other *SensorButtons* or to a desktop computer. The lower board contains the digital part with a 16-bit, low-power MSP430F1611 microcontroller running at up to 8 MHz. It provides 4kB of RAM, 48 kB of ROM and offers 4 power modes. The MSP430F1611 contains a 8-channel ADC to sample the sensor data from the upper board. The hybrid power supply circuitry is located on the lower board as well. It allows to power the *SensorButton* via a lithium-ion battery in combination with a solar cell. The voltages of the battery as well as the solar cell can be monitored via the microcontroller ADC to perform a maximum power point tracking [4].

2.1 Wireless Network

The *SensorButton* is intended to be used on the body to compute context information. Its wireless link transmits the results to other devices on the body or in the environment. To get an idea of where the receiving devices can be located, we have evaluated the communication range of the *SensorButton* on the body and its environment in an office environment. Table 1 shows our results for packet success rates from different parts of the body to others. Packets are counted as successfully transmitted if they pass the CRC test in the nRF2401 transceiver. Long range transmissions in free line of sight are possible with 63.93% success rate up to 50m. However, from about 20m distance, any object blocking the direct line of sight lets the reception rate drop to almost zero. Reliable transmissions is thus only possible in very short range. Objects blocking the direct line of sight line greatly reduce the transmission rate. Consequently the transmission range from the waist of a sitting person to the ankle when sitting consequently falls to 78.39% compared to a standing person. Such rates prohibit the communication of raw, unprocessed sensor data for an accurate activity recognition. A better approach is to preprocess data on the *SensorButton* itself, and only to transmit intermediate or finalized classification results, which typically have a smaller data volume as the complete sensor readings. Reducing the message count to be transmitted saves power, as the major power consumer on wireless sensor nodes are the transceivers.

When a sensor node takes part in a wireless network, it



Source	Destination	Rate
standing		
shoulder (A)	abdomen (C)	95.69 %
back (D)	abdomen (C)	97.12 %
ankle (F)	moving arm (E)	94.74 %
ankle (F)	still arm (E)	96.86 %
sitting		
chest (B)	wrist (E)	99.44 %
abdomen (C)	ankle (F)	78.39 %
Free line of sight		
4m		99.85 %
10m		99.64 %
20m		99.40 %
30m		97.15 %
40m		97.07 %
50m		63.93 %

Table 1: Transmission rates at the highest sending power level (0dBm), measured in the hallway of our office

has to follow various other tasks besides computing context information. Such tasks can be following the various network protocols used (MAC, clustering, routing...). Additionally, in real-life many different types of sensor nodes will be available in the environment, and some abstraction of the hardware is needed to allow our software to be ported to these different platforms. To integrate our sensor nodes into a network, we thus need an operating system. We have looked at different systems, such as TinyOS, Contiki [7], SOS [10], and others. Out of these we have chosen TinyOS for the reasons stated in the following section.

2.2 TinyOS

A major restriction for operating systems for wireless sensor network nodes are the hard restrictions on computational resources and power. An operating system for such nodes has to be extremely light-weight and has to incorporate power-saving modes into its operation. TinyOS is targeted to such

systems and provides the functionality needed. It is programmed in the nesC [9] C dialect, which allows a modular composition of the operating system components. The compiler only loads modules, which are actually needed. The peculiarities of programming in TinyOS have been analyzed in [8]. TinyOS provides a non-preemptive scheduler which allows to run different processes, or *tasks* sequentially.
Some of the advantages of using TinyOS as the operating system are:

- TinyOS has been ported to different platforms, such as Mica [11], Telos [21], TinyNode [6] and many others. This brings the advantage that the code can be ported with reduced effort.
- The modularity of TinyOS allows a flexible composition of the recognition algorithms. This simplifies the reuse of algorithm components and makes code also easier to understand.
- TinyOS has been used by many research groups and in industry and provides a large code base that can also be accessed for different projects. Thus implementation time is reduced by using such components.

We have ported TinyOS 2.0 to the *SensorButton*. The microcontroller is the same as one of the prime platforms for TinyOS, the Telos motes, which simplified the porting of the operating system, as only the external interfaces had to be adapted to the different hardware. We had to extend the ADC driver module for our purposes in two ways: 1) consecutive reading of the ADC channels, such that the 3 channels of the accelerometer are sampled as synchronous as possible. 2) DMA controlled sampling to achieve a 4kHz reading of the microphone without burdening the microcontroller too much. The major porting work was writing a driver for the nRF2401 transceiver, which was not yet supported by TinyOS. The TinyOS hardware abstraction made the use of the *SensorButton*-specific transceiver transparent to the application running on TinyOS.
The code for the activity recognition algorithms could be integrated by writing wrapper modules around the function calls. To allow the TinyOS scheduler to run other tasks during the classifier runtime, the classifier code had to be split into a sequence of tasks to be executed. This allows other tasks, such as network protocols, to be run in parallel by TinyOS, but adds additional overhead to the execution of the algorithm.

3. ACTIVITY RECOGNITION

The main challenge on performing online activity recognition on a platform like the *SensorButton* is to select algorithms which can achieve acceptable recognition performance with limited computation resources. Our methodology was to select a scenario with a number of activities to be recognized. These activities would then be recorded on multiple subjects for an offline analysis to determine the optimal settings for an online activity recognition algorithm running on the *SensorButton*.
We have chosen an office worker scenario and selected the 7 activities listed in table 2 to be recognized. The data processing flow used for our algorithms is as follows: The sensor data is segmented using a sliding window. The second step is to compute features that characterize the signal captured in the window, which are then fed into a classifier. In a last step, the result is communicated to other sensor nodes.

Drinking water
Moving a computer mouse
Writing on a whiteboard
Opening a drawer
Opening a cupboard
Typing on a keyboard
Writing with a pen

Table 2: Office worker activities to be recognized by the activity recognition algorithm

Sensors	**Accelerometer, light sensor**, microphone
Segmentation	Sample frequency (**32Hz**), window size (**2.5 sec**), window overlap (**70%**)
Features	**mean, energy, variance**, std. deviation, fluctuation, mean gradient, abs. gradient, mean crossing rate
Classification	**J48/C4.5, k-Nearest Neighbors (k-NN)**, Naïve Bayes, Bayes Net

Table 3: Design space for the activity recognition algorithm. Our choices for the actual implementation are marked in bold

Table 3 shows the design space of the activity recognition algorithm optimizations. To assess the influences of the different parameters, we have asked 9 test subjects to perform each activity 30 times. The sensor values have been collected with 4086 Hz from the microphone and with 108 Hz for the other two sensors. It served as the basis for an offline-analysis of the different parameters. The main goals of the optimizations were to achieve a high recognition rate with low processing complexity. In the following, we present the results we have found for each step of the algorithm.
Sensors: The microphone information is ruled out as the activities in the given scenario are dominated by motions and the accelerometer readings yield much more valuable information.
Segmentation: Figure 3 shows the effects of the window size and overlap on the performance of the classificators exemplarily for the k-NN classifier. A higher overlap improves the recognition performance, which leads to a trade-off to additional computational work needed to process all data multiple times. The size of the window needs to be chosen more carefully, since a too long as well as a too short window will degrade the performance of the classifiers. The sampling frequency also plays a major role in the resulting processing needed for the data. We have analyzed its effects by resampling our recorded data and have chosen a sampling frequency of 32 Hz. Lower sampling frequencies will significantly decrease the recognition rate. A window size of 2.5 seconds and an overlap of 70% resulted in a good compromise between processing needs and recognition performance.
Feature Extraction: Features can be ranked based on the information gain they provide to the classifier [14]. The fea-

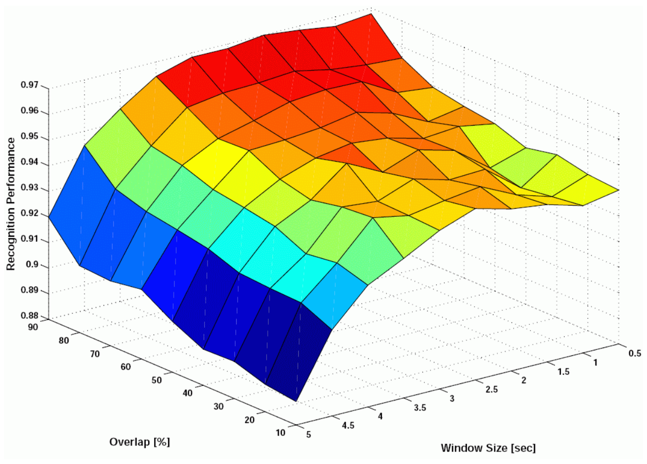


Figure 3: Evaluation of classification quality for online context recognition for the kNN classifier at 32 Hz

ture information gain must again be related to the cost, or computational complexity of computing it. Table 3 lists the evaluated features. The *mean*, *variance*, and *energy* values of the signal of the window have been selected for implementation and have been computed for all of the 3 accelerometer axes and the light sensor.

Classification: We have used the Weka Machine Learning Toolkit [26] to train the classifiers. All four classifiers we have used in our analysis showed similar recognition behavior. Thus the complexity of the calculation has been the decisive point, where we have chosen to use k-NN with a Manhattan Distance, and a J48/C4.5 decision tree, being the classifiers with the least complexities but rendering acceptable performance.

4. RESULTS

The selected k-NN and J48/C4.5 classifiers have been implemented for the *SensorButton* in two configurations: a custom code, platform-specific executable and an integration into TinyOS.

In the offline evaluation on a PC, the classifiers reached 98% recognition rate. The implementation for the *SensorButton* let the recognition rate drop to 86% for the J48/C4.5 and 91% for the k-NN (k=5) classifier. This is mainly due to the reduction of the bit accuracy from floating point to fixpoint, which is needed for an online implementation on a 16-bit microcontroller.

The custom code implementation has been compiled using the IAR C/C++ Compiler for MSP430 version 3.42A. TinyOS compiled with nesC 1.2.7b (gcc 3.4.4 for Cygwin). The memory footprints of the implementations are shown in table 4. TinyOS adds about 7kB of code to the application. This code contains the TinyOS scheduler, different hardware control and access modules, and wireless protocol implementations, which do not exist in the custom solution.

More important with respect to power consumption is the execution time of the algorithms, which is shown in table 5. The sampling time indicates the time needed to access the ADC to sample the sensors, which is done at 32Hz. The total time for each classifiers includes the time for sampling, feature calculation, and communication for 1 of consecu-

	TinyOS		Custom Code	
	k-NN	C4.5	k-NN	C4.5
Code (ROM)	33'428	9930	26'317	2978
Data (RAM)	2476	2476	2338	2338

Table 4: Code size in bytes of the implementations. The large code memory requirements for the k-NN derive from the feature space point database, which contains 1312 entries

tive classifications. It includes the sampling time 24 times, as with 70% window overlap, every sample in the 2.5 second sampling window is used about 3 times. A unexpected observation was that the gcc compiler optimized the same code for the feature calculation and the k-NN classifier better than the IAR compiler. This resulted in an overall lower computation time for the C4.5 classifer in TinyOS.

	TinyOS [ms]	Custom Code [ms]
Sampling	1.01	0.03
Features	7.2	27
k-NN	54	80
C4.5	0.02	0.02
Communication	22	10
Total kNN	107.44	117.72
Total C4.5	53.46	37.74

Table 5: Execution Times on custom and TinyOS code for 80 samples and 70% window overlap including sampling interruptions at 32 Hz

To get the power values, the computation times have to be multiplied with the power consumption in the different states. Table 6 shows the power consumption of the *SensorButton* when different modules are enabled. It shows the large power drawn by the microphone and identifies the wireless link as the biggest power consumer. The power needed by the transceiver can be reduced using a duty-cycling MAC protocol [5], which allows to turn it off for certain times. If the *SensorButton* does not have to receive any data at all, the power requirements can further be reduced by only turning the radio on when it is needed to send a message containing the recognition result. The same applies for the sampling of the sensors, which could be turned off when not in use.

If the *SensorButton* is always driven in the optimal power

Active hardware modules	mW
MSP430 only (at 4MHz)	0.99
plus light sensor and accelerometer	4.78
plus Microphone	17.63
plus nRF2401	28.08

Table 6: Power consumption of the *SensorButton* in different operation modes

mode, meaning sensors and transceivers are only turned on when actually needed, a theoretical minimum energy requirement per classification of 803 μJ for k-NN and 749 μJ for the C4.5 can be computed for the TinyOS implementation. However, this is a theoretical value, as sensors as well as the transceiver need a startup time before they can be used. Our TinyOS implementation leaves sensors on all the time and enables the transceiver only for transmission. This renders an average power consumption of 5.46 mW when communicating 2 classifications per second. Using our 150 mAh battery, this configuration can be run for 75 hours.

5. COLLABORATIVE CONTEXT RECOGNITION

A collaborative context recognition of multiple sensor network nodes may be beneficial for different tasks; to recognize whether two devices are on the same person [18], the device position on the body [16], or of course the activity of a person using multiple sensors on the body [2]. The classification results from different devices can also be communicated to a *meta-classifier*. It fuses the different results to improve the overall classification result [22].

In a dynamic environment such as an on-body-network, it is difficult to make assumptions about what devices are available in the environment beforehand, and must thus be assessed during the runtime of the network. As people move around, the situation can also be changing quickly, and distributed context algorithms must be constantly adapted to the momentary configuration of the wireless network.

Titan [19] is a framework designed for this task. It dynamically reconfigures wireless sensor networks to perform algorithms that have previously been specified by designers as *task graphs*, describing the data flow from sensors to recognition result. These algorithm task graphs are loaded into a database in the wireless network. When a node calls for the execution of a task graph, *Titan* examines the currently available sensor network for its capabilities and starts a distributed task graph execution.

The task graph remains executing until network nodes fail or get out of reach. In this case, *Titan* computes a new network configuration and dynamically reconfigures single network notes, such that the algorithm can continue to run. *Titan* has the following distinct advantages:

- Ease of use – Algorithms are designed as task graphs. All communication between those tasks are taken care of by *Titan*.
- Portability – *Titan* is based on TinyOS, which has been ported to different platforms. Due to the abstraction from the actual hardware it can run on heterogeneous networks.
- Flexibility – It adapts to the dynamic wireless sensor network configuration by reconfiguring individual sensor nodes to keep the distributed algorithms running.
- Speed – *Titan* can reconfigure a sensor node in less than 1 ms [19].

The activity recognition algorithms presented in section 3 have now been incorporated into *Titan* and enable it to perform more complex activity recognitions.

6. DISCUSSION

Task graphs for context recognition has been employed by the *Context Recognition Network Toolbox* [1] in different projects [25]. It is intended to be used on a Linux-based computer. Its ease of use has convinced us to bring the same algorithm design principle to Wireless Sensor Networks. For this purpose we had to add the capability to handle heterogeneous and dynamically organized wireless sensor networks, which have done in *Titan*.

Capturing activity information by accelerometers has been done by different groups [2, 13, 17]. However, these approaches require more processing power as miniaturized devices, such as the *SensorButton* can provide. Initial work in the direction of context aware wristwatch sized computers has been done by [15], where simple activities such as walking, sitting, and standing have been recognized. The work at hand goes further in that it recognizes more complex activities.

In future work we will start making use of the classifiers ported to *Titan* and go into the recognition of activities using multiple sensors located on the body. This will start to rise a series of questions that we want to target, such as: how are the recognition algorithms influenced by unreliable communication links, ie. missing data? From the available sensors, which should be selected for the actual context recognition, and if they fail, which should replace them? Another interesting question is how to derive, which sensors are all on the same body, and clustering them such as to know which sensors will presumably keep together for some time.

7. CONCLUSION

The *SensorButton* platform has been presented and an activity recognition algorithm has been developed and optimized for an online activity recognition in an office worker scenario. This algorithm has been implemented twice, once as a single application and a second time integrated in the TinyOS operating system. A surprising result is that the implementation in TinyOS executes faster, which must be due to the different compilers used. The TinyOS implementation also uses about 6kB more code memory, while the dynamic memory requirements are comparable. The implementations are able to perform up to 18 context evaluations per second.

The activity recognition algorithms can be integrated in a dynamic wireless sensor network using *Titan*. In this framework, the algorithms adapt dynamically to the available sensor network nodes.

8. ACKNOWLEDGMENTS

This paper describes work undertaken in the context of the e-SENSE project, ‘Capturing Ambient Intelligence for Mobile Communications through Wireless Sensor Networks’ (www.ist-e-SENSE.org). e-SENSE is an Integrated Project (IP) supported by the European 6th Framework Programme, contract number: 027227

We would like to thank Andreas Bubenhofer and Christoph Walser for their valuable help in bringing TinyOS to the *SensorButton*.

9. REFERENCES

[1] D. Bannach, K. Kunze, P. Lukowicz, and O. Amft. Distributed Modular Toolbox for Multi-modal Context Recognition. In *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS)*, pages 99–113, March 2006.

[2] L. Bao and S. S. Intille. Activity Recognition from User-Annotated Acceleration Data. In *Proceedings of*

the 2nd International Conference on Pervasive Computing (PERVASIVE), pages 1–17, 2004.

[3] N. B. Bharatula, M. Stäger, P. Lukowicz, and G. Tröster. Empirical Study of Design Choices in Multi-Sensor Context Recognition Systems. In *Proceedings of the 2nd International Forum on Applied Wearable Computing (IFAWC)*, pages 79–93, Mar. 2005.

[4] N. B. Bharatula, J. A. Ward, P. Lukowicz, and G. Tröster. Maximum-Power-Point Tracking for On-Body Context Systems. In *Proceedings of the IEEE International Symposium on Wearable Computers (ISWC)*, 2006.

[5] I. Demirkol, C. Ersoy, and F. Alagöz. MAC Protocols for Wireless Sensor Networks: A Survey. *IEEE Communications Magazine*, 2005.

[6] H. Dubois-Ferrière, R. Meier, L. Fabre, and P. Metrailler. TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 358–365, 2006.

[7] A. Dunkels, B. Gronvall, and T. Voigt. Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the IEEE International Conference on Local Computer Networks (LCN)*, pages 445–462, 2004.

[8] D. Gay, P. Levis, and D. Culler. Software Design Patterns for TinyOS. In *Proceedings of the 2005 ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 40–49, New York, NY, USA, 2005. ACM Press.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (SIGPLAN)*, pages 1–11, 2003.

[10] C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A Dynamic Operating System for Sensor Nodes. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, pages 163–176, 2005.

[11] J. Hill and D. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, 2002.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[13] T. Huynh and B. Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 159–163, 2005.

[14] H. Junker. *Human Activity Recognition and Gesture Spotting with Body-Worn Sensors.* Dissertation, ETH Zurich, 2005.

[15] A. Krause, M. Ihmig, E. Rankin, D. Leong, S. Gupta, D. Siewiorek, A. Smailagic, M. Deisher, and uttam Sengupta. Trading Off Prediction Accuracy and Power Consumption for Context-Aware Wearable Computing. In *Proceedings of the 9th IEEE International Symposium on Wearable Computers (ISWC)*, pages 22–26, 2005.

[16] K. Kunze, P. Lukowicz, H. Junker, and G. Tröster. Where am I: Recognizing On-Body Positions of Wearable Sensors. In *International Workshop on Location- and Context-Awareness (LoCa)*, pages 264–275, May 2005.

[17] J. Lester, T. Choudhury, and G. Borriello. A Practical Approach to Recognizing Physical Activities. In *International Conference on Pervasive Computing (PERVASIVE)*, pages 1–16, 2006.

[18] J. Lester, B. Hannaford, and G. Borriello. "Are You with Me?" – Using Accelerometers to Determine If Two Devices Are Carried by the Same Person. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE)*, pages 33–50, 2004.

[19] C. Lombriser, M. Stäger, D. Roggen, and G. Tröster. Titan: A Tiny Task Network for Dynamically Reconfigurable Heterogeneous Sensor Networks. In *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, pages 127–138, 2007.

[20] P. Lukowicz, J. A. Ward, H. Junker, and G. Tröster. Recognizing workshop activity using body worn microphones and accelerometers. In *Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE)*, pages 18–32, 2004.

[21] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, page 48, 2005.

[22] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman. Activity Recognition from Accelerometer Data. In *Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 2005.

[23] D. Roggen, N. B. Bharatula, M. Stäger, P. Lukowicz, and G. Tröster. From Sensors to Miniature Networked SensorButtons. In *Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS06)*, 2006.

[24] M. Stäger, P. Lukowicz, and G. Tröster. Implementation and Evaluation of a Low-Power Sound-Based User Activity Recognition System. In *Proceedings of the 8th IEEE International Symposium on Wearable Computers (ISWC)*, pages 138–141, Nov. 2004.

[25] T. Stiefmeier and C. Lombriser and D. Roggen and H. Junker and G. Ogris and G. Tröster. Event-Based Activity Tracking in Work Environments. In *Proceedings of the 3rd International Forum on Applied Wearable Computing (IFAWC)*, Mar. 2006.

[26] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kauffman Publishers Inc., 2000.