Rapid Prototyping of Activity Recognition Applications

The CRN Toolbox enables fast implementation of activity and context recognition systems, featuring mechanisms for distributed processing and support for mobile and wearable devices.

David Bannach and Paul Lukowicz
University of Passau

Oliver Amft
ETH Zurich

Today, the development of activity recognition systems has two main phases. The first phase is to design the recognition method—the sensor setup, feature set, classifiers, classifier parameters, fusion methods, and so on. Here, designers feed experimental data offline into conventional rapid-prototyping tools such as Matlab. These tools provide a rich reservoir of off-the-shelf, parameterizable algorithms and visualization methods, which enable the testing of different system variants quickly without time-consuming implementation work.

Unfortunately, most of these simulation environments aren't suitable for actually running applications, especially in mobile and pervasive environments. They typically depend on custom engines or libraries requiring large memory footprints and high computing power. Consequently, a separate, second phase is usually necessary to implement activity recognition applications. This phase implements the selected algorithms in an appropriate programming language and then distributes them to specific devices. Relevant issues in this phase include sensor interfaces, synchronization of the sensor signals, and optimization for specific devices (for example, floating-point or fixed-point calculation).

The Context Recognition Network (CRN) Toolbox (http://crnt.sf.net) combines these two phases and permits quick construction of complex multimodal context recognition systems for immediate deployment in the targeted environment. We developed the CRN Toolbox to ease the process of building activity recognition systems. Three case studies demonstrate the versatility of the CRN Toolbox. In these case studies, we deployed the CRN Toolbox to support information flow in hospitals, monitor walking habits to help prevent cardiovascular diseases, and recognize hand gestures in a car-parking game. The spectrum of implemented solutions indicates that our approach is viable in the diverse environments of wearable and server-based applications.

Comparing the CRN Toolbox with other tools

The CRN Toolbox isn't general-purpose pervasive middleware such as RUNES (Reconfigurable Ubiquitous Networked Embedded Systems),[1] nor a sensor-node operating system such as TinyOS.[2] Neither is it a high-level framework for rule-based automation, such as VisualRDK.[3] Rather, it's a tool set specifically opti-

Published by the IEEE CS ■ 1536-1268/08/$25.00 © 2008 IEEE

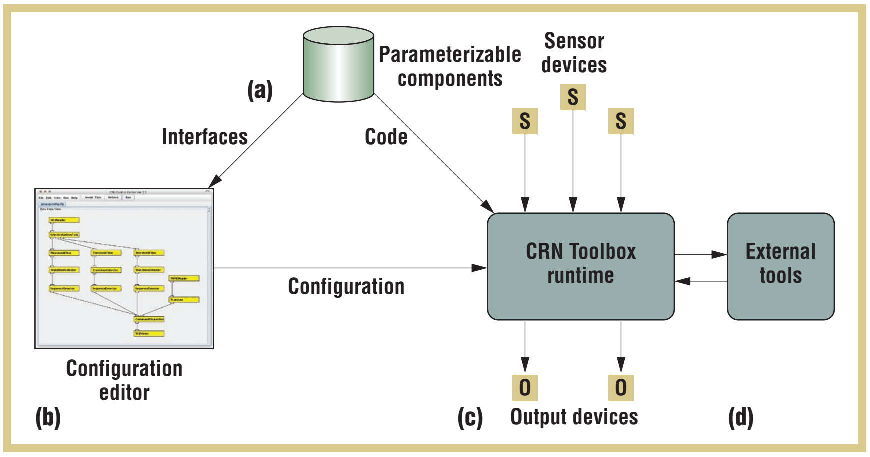


Figure 1. Concept of the Context Recognition Network (CRN) Toolbox: (a) repository of parameterizable software components, including I/O device readers and writers, filtering and classification algorithms, and components for splitting, merging, and synchronizing data streams; (b) graphical editor for specifying data flow and configuring components; (c) the CRN Toolbox runtime environment for online execution of the configured software components; and (d) arbitrary external tools—for example, live data-stream plotting or another CRN Toolbox (local or remote)—communicating with the Toolbox runtime.

mized for implementing multimodal, distributed activity and context recognition systems running on Posix operating systems. Like conventional rapid-prototyping tools, the CRN Toolbox contains a collection of ready-to-use algorithms (signal processing, pattern classification, and so on). Unlike classic event detection in homogeneous sensor networks—for example, DSWare (Data Service Middleware)[4]—it supports complex activity detection from heterogeneous sensors. Its implementation is particularly optimized for mobile devices. This includes the ability to execute algorithms, whether in floating-point or fixed-point arithmetic, without recoding. Moreover, with its mature functionality, the CRN Toolbox isn't likely to suffer from limited user acceptance as the Context toolkit framework did.[5]

The CRN Toolbox contains dedicated building blocks for interfacing a broad range of sensor nodes. It also supports synchronization, merging, and splitting of data streams. In contrast to the PCOM (Pervasive Computing Component System) model,[6] which focuses on contract-based spontaneous configuration, the Toolbox relies on a known network topology. Users can flexibly distribute applications among devices (including servers) by simply starting the configured Toolbox runtime on the appropriate system. Another important feature is the ability to interface conventional simulation environments such as WEKA (Waikato Environment for Knowledge Analysis, www.cs.waikato.ac.nz/~ml). The functionality is accessible through a graphical configuration editor, which enables constructing complex applications by connecting and configuring a set of task icons corresponding to different processing steps.

The concepts the CRN Toolbox uses—graphical programming, data-driven computation, parameterizable libraries, and distribution—are themselves not new. But the CRN Toolbox has optimally adapted and integrated these concepts for rapid, efficient implementation of context recognition systems.

Toolbox concept

The concept of the CRN Toolbox stems from the observation that most activity recognition systems rely on a relatively small set of algorithms. These include sliding-window signal partitioning, standard time and frequency domain features, classifiers, and time series or event-based modeling algorithms.

The key differences between systems involve sensor choice, parameterization of algorithms (for instance, sliding-window size), and data flow. The data flow can be as simple as feeding 1D sensor data to a mean filter and a classifier. This could be a configuration for recognizing sitting and standing from an upper-leg accelerometer, for example. It can be as complex as fusing data from tens of heterogeneous sensors, working with different sampling frequencies, different feature computations, and even different classifiers. In such complex systems, different platforms often handle different sensor subgroups—for example, certain mobile devices and servers with stationary sensors. The implementation must handle the distributed computation, collect the data, and synchronize the different data streams.

The CRN Toolbox simplifies the implementation of even complex, distributed context recognition systems to the following three steps:

- Compile the Toolbox for all platforms on which it needs to run.
- Select and configure the algorithms and data flow for each platform.
- Start the Toolbox on each platform with the dedicated configuration.

If it's necessary to analyze algorithms that aren't presently available in the CRN Toolbox, users can easily interface rapid-prototyping tools running on a remote server.

Figure 1 shows an overview of the CRN Toolbox concept. The step-by-step configuration guide presents a simple example for recognizing kitchen activities from the user's on-body sensors.

TABLE 1
Summary of tasks currently provided by the CRN Toolbox.

Task category (no. of tasks)	Task implementations
Generic reader (4)	Reading from file, keyboard, TCP socket, or serial device (including Bluetooth), using a decoder plug-in
Specific reader (18)	ADS* (heart rate), ARSB (walking sensing), BTnode, Hexamite, ID-10 RFID, Lukotronic, NMEA (GPS), MyHeart protocol, SkyeTek M1-mini RFID, Tmote force-sensing resistors, Tmote RFID, Tmote magnetic distance, TMSI fiber protocol, Suunto ANT protocol, Web interface input, Xsens MT9/MTi, Xsens Xbus, Wii Remote
Channel reordering (4)	ChannelSelect, SelectiveSplitterTask, SimpleMerger, SyncMerger
Filtering (4)	FilterTask, TransitionDetector, VecLen, Einsnorm
Filter plug-ins (16)	Average signal energy, band energy ratio, bandwidth, center of gravity, entropy, FFT, fluctuation, peak, max, mean, median, slope, scale, spectral roll-off frequency, threshold, variance
Classification tasks (9)	Distance2Position, Hexamite2D, HMMs, KNN, PCFG parser, RangeChecker, SequenceDetector, SimpleHexSensClassification
Miscellaneous (4)	Synchronizer, Heartbeat, Valve, Nothing
Writer (9)	TCP server, TCP client, serial port, file, console, MyHeart protocol, graph display, image display, Nirvana (silent sink)
Encoder plug-ins (9)	ARFFEncoder (WEKA), BinaryEncoder, CmdEncoder, IntLinesEncoder, JSONEncoder, PlottingEncoder, TextLabelEncoder, TimestampedLinesEncoder, SuperPacketEncoder
Decoder plug-ins (4)	ASCIIDecoder, FloatLinesDecoder, IntLinesDecoder, StringLinesDecoder

* ADS: Advanced Digital Strap (Philips heart rate belt); ARFF: Attribute-Relation File Format (WEKA); ARSB: activity recognition sensor board; FFT: fast Fourier transform; HMM: hidden Markov model; KNN: k-nearest neighbor; PCFG: probabilistic context-free grammars; WEKA: Waikato Environment for Knowledge Analysis

Reusable components

The basic building blocks provided by the CRN Toolbox are the reusable, parameterizable components. Conceptually, the components are active objects that operate on data streams. We refer to them as *tasks*. They encapsulate algorithms and data, and they each have an individual thread of execution. In essence, tasks run in parallel, waiting for *data packets* to arrive at their *in-port*. They then process the packet's payload according to their algorithm and parameter settings, and provide the modified data packet at their *out-port*. Depending on the configured data flow, subsequent tasks will receive the packet for further processing.

The Toolbox provides *reader* and *writer* tasks for interfacing with input and output devices, processing algorithms for data filtering and classification, and components for splitting, merging, and synchronizing data streams. Table 1 summarizes currently available tasks. Detailed task descriptions are available as help pages. The list is constantly growing as increasingly more users contribute to the project. Numbers in parentheses represent the number of tasks in each category.

Every task has an individual number of parameters that control its operation. For example, the k-nearest neighbor (KNN) classifier task uses the k, a file name with training data, and an optional step-size parameter.

The encapsulation in active objects and the parameterization proved essential for reusing the actual code. So, for most applications, the fact that the Toolbox is implemented in C++ is insignificant, yet those applications benefit from the efficient runtime.

The motor: Runtime environment and flow control

The Toolbox runtime provides the vital environment for tasks to operate. It handles dynamic creation and configuration of tasks as well as configuration of the data flow.

For parameter handling, the Toolbox uses the JavaScript Object Notation (JSON) format, with an object loader in the "get instance by name" style.[7] Thus, users can configure the Toolbox at runtime through text-based configuration files that define settings for tasks and the data flow that the application needs.

Directed *connections* from out-ports to in-ports specify the data flow between tasks. Each data packet transmitted along these connections contains data entities belonging to one time instant. A packet's payload is organized as a vector of values from an abstract data type. Moreover, the packets contain a time stamp and sequence number. For combining multiple streams, the Toolbox provides *merger* tasks. Mergers combine the payloads of packets from separate in-ports and synchronize data streams with different sampling rates.

We used pointer references to pass data packets along the internal connections through the task network. Packets are cloned only if more than one receiver connects to the same out-port. This implementation of the runtime core ensures high packet-processing performance. Moreover, we preserved processing performance by providing operations to the task developer that, like the += operator,

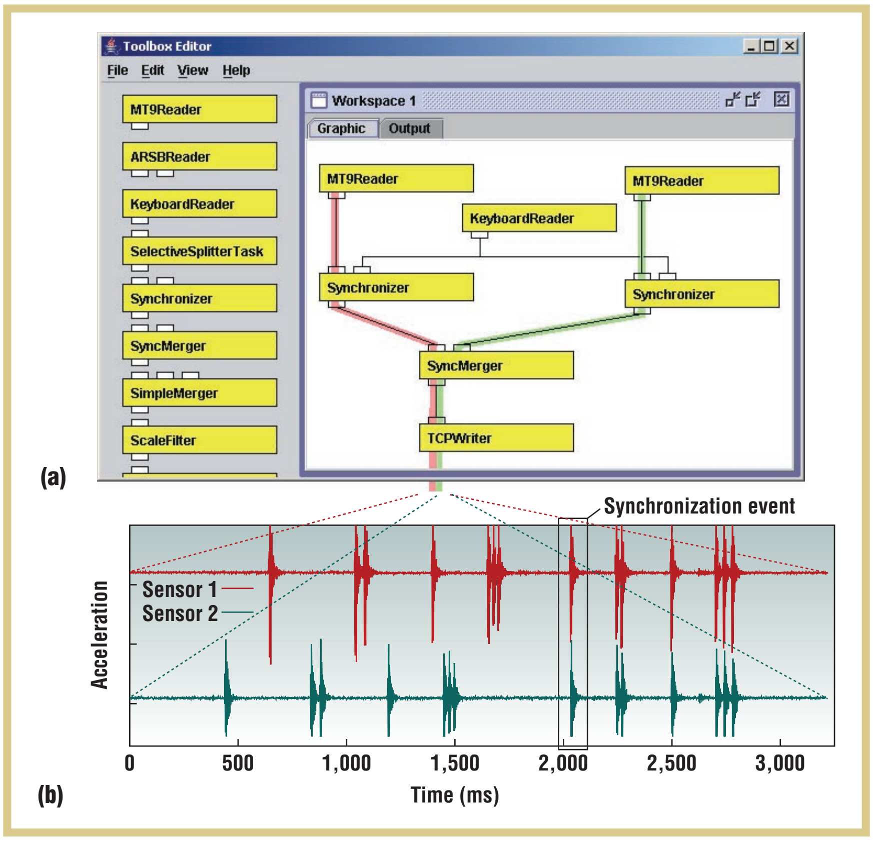


Figure 2. Example using two Xsens MT9 acceleration sensors: (a) CRN Toolbox graphical configuration editor with synchronization setup; (b) data alignment achieved at an event detected by the Synchronizers.

inherently modify data objects instead of allocating new objects.

Synchronizing independent data streams

Synchronization of the data streams from different sensors is a major issue in multimodal activity recognition. When using several independent sensors, it's important to synchronize their data streams to a common starting point.

A feasible concept for this type of synchronization is aligning streams on events recorded by all sensors simultaneously—for example, a user jumping up with a set of on-body acceleration sensors. We implemented this concept in the Synchronizer and SyncMerger tasks. Figure 2 depicts the solution for the example of two Xsens MT9 acceleration sensors. The jump inserted a characteristically high acceleration amplitude. The Synchronizer tasks detect the peaks caused by these events and adjust data packet time stamps accordingly. The SyncMerger combines the data streams by aligning the time stamps. The Synchronizer tasks are manually activated—for instance, through a KeyboardReader—to limit the alignment phases to controlled time frames. Our initial analysis of the method showed that an alignment of 0.5 seconds and better was possible.

Readers: Sensor hardware encapsulation

The CRN Toolbox implements sensor interfaces as tasks without in-ports. These *reader* tasks instantiate new data packets for data samples acquired from sensors (or other sources) and provide these packets on their out-port. Our architecture supports various reader implementations that can capture different sensors or other sources, such as web pages, application outputs, and data files.

For activity annotation, we implemented a keyboard reader to perform online labeling of data. This reader proved very helpful, because it enables storing the labels with the raw data for later evaluation.

Writers: Communication for distributed processing

Writer tasks are the key to distributed execution and use of external tools. They forward data received at their in-port to external interfaces (such as files, displays, or network connections). For network connections, we use TCPWriter and TCPReader tasks to communicate via TCP/IP sockets. The CRN Toolbox transmits data packets on the channel in a serialized form. The Toolbox obtains the serialization from an encoder plug-in in the TCPWriter task. Similarly, the TCPReader uses a decoder plug-in for deserialization. Thus, two CRN Toolboxes running independently—for example, on different hosts—can collaborate using the writer-reader communication.

Using this mechanism, the Toolbox can link to arbitrary programs based on compatible interfaces. Currently, such interfaces exist for Matlab and WEKA, both of which support data visualization and pattern recognition in experiments and demonstrators.

Easy configuration

The Toolbox's rapid-prototyping capabilities increased our need for an easy, quick configuration editor. Figure 2 shows the graphical configuration editor. Users can drag tasks from a library into the workspace and connect them to other tasks with just a few mouse clicks. The Java-based editor produces configuration files for the Toolbox. (See the "How to Cook: A Step-by-Step Guide" sidebar for an example of how easy it is to build activity recognition applications with the CRN Toolbox.)

Case studies

The vitality of a framework such as

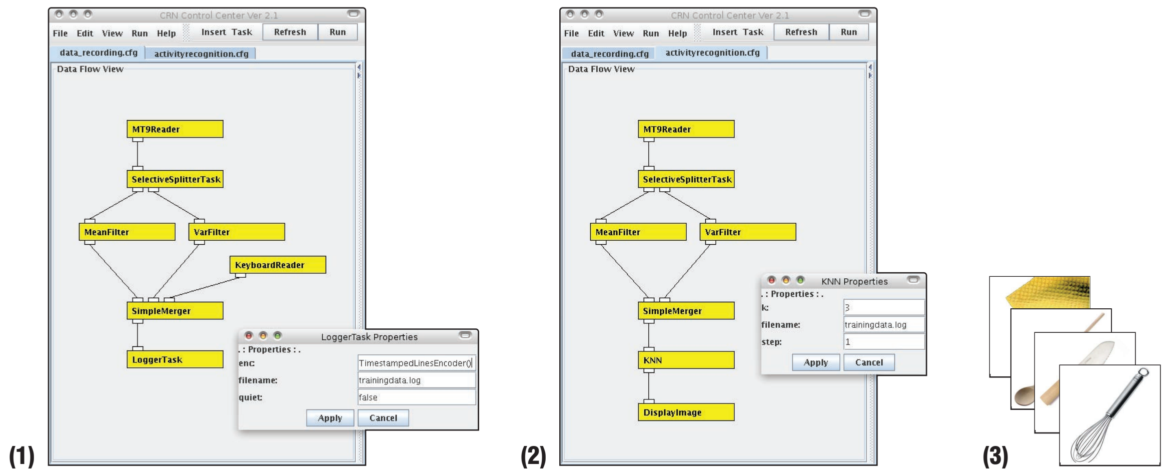
How to Cook: A Step-by-Step Guide

The CRN Toolbox makes building activity recognition applications easy. For example, implementing your own kitchen activity recognition takes only five steps, including classifier training. Moreover, you don't have to write additional code.

The ingredients are a motion sensor mounted on a glove, a wearable computer or "kitchen PC," and the CRN Toolbox. In this guide, we use the MT9 sensor from Xsens. Typical activities include stirring, whisking, cutting bread, slicing onions, and wiping with a cloth.

1. Using the graphical configuration editor, create a configuration for recording training data (see figure A1). Begin by adding the MT9Reader to acquire data from the MT9 sensor at 100 Hz, and provide all nine channels on its out-port. Use SelectiveSplitterTask to choose the channels of interest, and then send them to MeanFilter and VarFilter (variance). Set the sliding window size to 100 (1 second). Use SimpleMerger to combine the two data streams again, and add the output of a KeyboardReader task. This task annotates the recording by keystrokes. Finally, add LoggerTask to write the resultant data streams into a file.
2. Select an annotation key for training each activity. Connect the sensor and start the Toolbox with the created configuration. Then, wearing the sensor glove, perform each activity for about 30 seconds. At the beginning of each activity, press its selected annotation key.
3. Review the recorded training data in the log file and reduce it to about 100 samples per activity class. The number in the last column of the log file indicates the class label.
4. Modify the first configuration to include the classifier and the output task (see figure A2). Remove KeyboardReader, because from now on the classifier will do the annotation. Specify the file name of the training data in the properties of the KNN task. Attach the DisplayImage task to the KNN and specify the picture that should display on the screen for each recognized activity category.
5. Start the Toolbox with the new configuration. Now you can work in the kitchen as you wish and let the Toolbox track your activities or, even better, feed the results into a context-aware cookbook (see figure A3). Bon appétit!

To improve the system, you could add more sensing modalities (such as location), select useful features, or use more sophisticated recognition algorithms.



. : Properties : .
enc: TimestampedLinesEncoder()
filename: trainingdata.log
quiet: false

. : Properties : .
k: 3
filename: trainingdata.log
step: 1

Figure A. Configurations for kitchen activity recognition: (1) recording of training data; (2) online classification and display of results; and (3) example output of the classification using DisplayImage.

the CRN Toolbox stems from its continual development and deployment in various projects. The showcase of applications in industry projects, student classes, and demonstrators (see tables 2 through 4) highlights the CRN Toolbox's maturity and widespread use. These projects have successfully deployed the Toolbox on different platforms, including

- Linux running on arm32, i386, and amd64 systems;
- MacOSX running on i386s and iPhones; and
- Cygwin running on i386s.

Here, we depict three case studies

TABLE 2
Major industry projects using the CRN Toolbox.

Project description	Uses and deployment of CRN Toolbox
WearIT@Work: supports hospital information flow, RFID for patient identification, gesture-controlled access to patient documents using wrist-worn motion sensor.[8,9]	• Data capture, gesture recognition, control of hospital's document browser. • Several demonstrators and test systems were built and a hospital trial was conducted. • Platform: Q-Belt Integrated Computer (QBIC), Linux on arm32.
WearIT@Work production support: activity recognition of car assembly and maintenance,[10] uses inertial motion and indoor location sensors.	• Recording multimodal sensor data: Xsens, Hexamite, ultrasound, muscle force; various demonstrators. • Platform: Linux on i386.
MonAMI dynamic monitoring services.	• Dynamic reconfiguration of the Toolbox, depending on available sensors and registered services. • Platforms: Linux on i386 and arm32.
MyHeart walking habits: online classification of walking activities and intensities to support active lifestyle and improve fitness.	• Acquisition of heart rate, acceleration, and air pressure classification. • Streaming results to a mobile phone and professional coaching center. • Platform: Linux on arm32 (QBIC).
Location tracking: GPS-based local map visualization.	• GPS position logging (NMEA protocol) and conversion for dynamic map display, forwarding to central mission server. • Platform: Linux on arm32 (QBIC).

TABLE 3
Student class projects using the CRN Toolbox.

Project description	Uses and deployment of CRN Toolbox
ISWC 06 tutorial "Hands-on Activity Context Recognition": building a gesture recognition system for controlling simulated car-parking game with real waving gestures; 12 participants.	• Testing algorithms and gesture types using simulated data streams from a motion sensor glove. • 9 components, 7,000 LOC, Linux amd64 platform.
Number-entering game: entering binary digits using a motion sensor only; practical exercise and competition for ambient-intelligence lecture; 15 students in 5th semester.	• Understanding algorithms, modularization, and interfacing to sensor data. • 7 components, 2,000 LOC, Linux i386 platform.
Location estimation and activity recognition: ultrasonic and motion sensors; practical exercise for ambient intelligence lecture; 12 students in 5th semester.	• Understanding algorithms and location-tracking challenges. • 7 components, 2,000 LOC, Linux i386 platform.
Interactive World project: software project to implement gesture control for games (Pong, Tetris), three days, 19 students in 4th semester.	• Implementing TCP reader task, using KNN classifier (gesture recognition). • 5 components, 1,200 LOC, Linux i386 platform.
Activity monitoring: training a classifier to recognize human activities (sitting, standing, walking, running) and visualizing results; 10 students in 4th semester.	• Learning a classifier's concepts and operation, implementation, and testing. • 7 components, 2,000 LOC, Linux i386 platform.

from different areas, outlining the use of the CRN Toolbox.

Supporting information flow in hospitals

Along with our clinical partners in the EU-sponsored WearIT@Work project, we developed a solution to improve information flow for the hospital ward.[8,9] During the ward rounds, doctors determine patients' further treatment under tight time limitations. Access to patient documents at bedside would enable doctors to make decisions on the basis of all available information. Notebooks or PCs are impractical for this task because their operation is time consuming and distracting and involves touching unsterilized devices while in contact with patients.

Our wearable solution simplifies document access. When the doctor comes to the patient's bed, the bedside monitor automatically displays a document list for that patient. The doctor can then browse the documents by pointing at the monitor and swiveling his or her forearm. The system the doctor wears consists of the QBIC (www.qbic.ethz.ch) running the CRN Toolbox, an Xsens motion sensor, and an RFID reader. The doctor wears the QBIC as a belt; the Xsens motion sensor and RFID reader attach to the lower

TABLE 4
Demonstrator projects using the CRN Toolbox.

Project description	Uses and deployment of CRN Toolbox
Parking game: controlling a virtual driver and car with real hand gestures in a parking game.[11]	• Capturing glove-based inertial sensor data, gesture spotting using explicit segmentation, gesture event search, and fusion steps. • Controlling the game visualization engine. • Platform: Linux amd64.
Dietary activity tracking using PCFG: inference of food intake cycles from activities.[12]	• Simulating activity event input, PCFG parsing, reporting results. • Platform: Linux amd64.
Hammering and screwdriving demo: recognizing assembly activities (hammering, screwdriving, sanding, sawing) with a motion sensor in a glove.	• Xsens motion sensor capturing, classifying activities, displaying recognition results on screen and via wireless connection. • Platform: Linux on arm32 (QBIC).

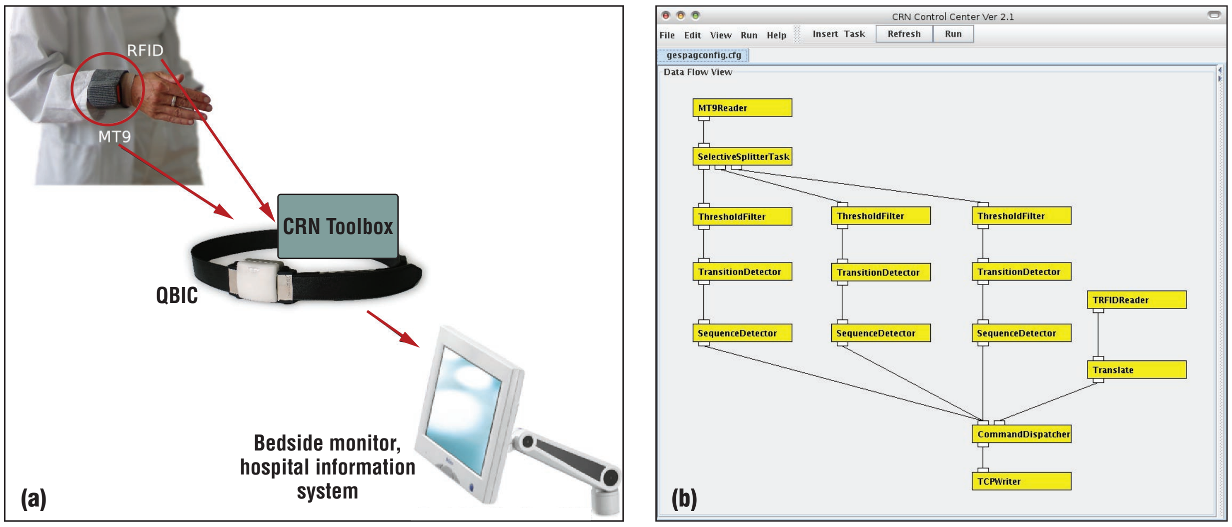


Figure 3. Hospital information flow example: (a) hospital information support system setup; (b) CRN Toolbox configuration.

arm. The patient wears an RFID tag. At each patient's bed, the bedside monitor displays documents from the hospital's information system.

Figure 3 shows the Toolbox configuration. In this configuration, we use threshold detection and sequence-matching tasks (ThresholdFilter, TransitionDetector, and SequenceDetector) to process each gyroscope axis of the motion sensor. This setup can detect forearm gesture sequences such as *swivel left, then right* (open-document command). The CommandDispatcher acts as a gateway, forwarding only those commands in active state (controlled by an activation gesture). This task also consumes the patient identification from RFID. Finally, the TCPClientWriter transmits the commands to the document browser of the hospital's information system.

We tested the complete setup in a two-week trial with doctors in an Austrian hospital. The system's shortcomings mainly concerned gesture detection robustness and sensor wearability, which we are now investigating.

Monitoring walking habits

With our industry partners in the EU-sponsored MyHeart project, we investigated new approaches for preventing cardiovascular diseases and maintaining low disease risks. Because many daily activities involve walking, we developed a walking-habits monitor that supports active walking and can track activity intensity.

In this setup, the QBIC serves as a central data acquisition and processing hub, running the CRN Toolbox. A custom sensing unit monitors user activity. This unit contains acceleration and air pressure sensors attached to the belt. Additionally, a heart rate chest belt uses Bluetooth to communicate with the QBIC. Based on features from the belt sensor unit, we classify walking straight, up, down, and idle as well as using the elevator up or down. The Toolbox forwards the results, along with the heart rate, to a mobile phone.

Figure 4 shows the final Toolbox con-

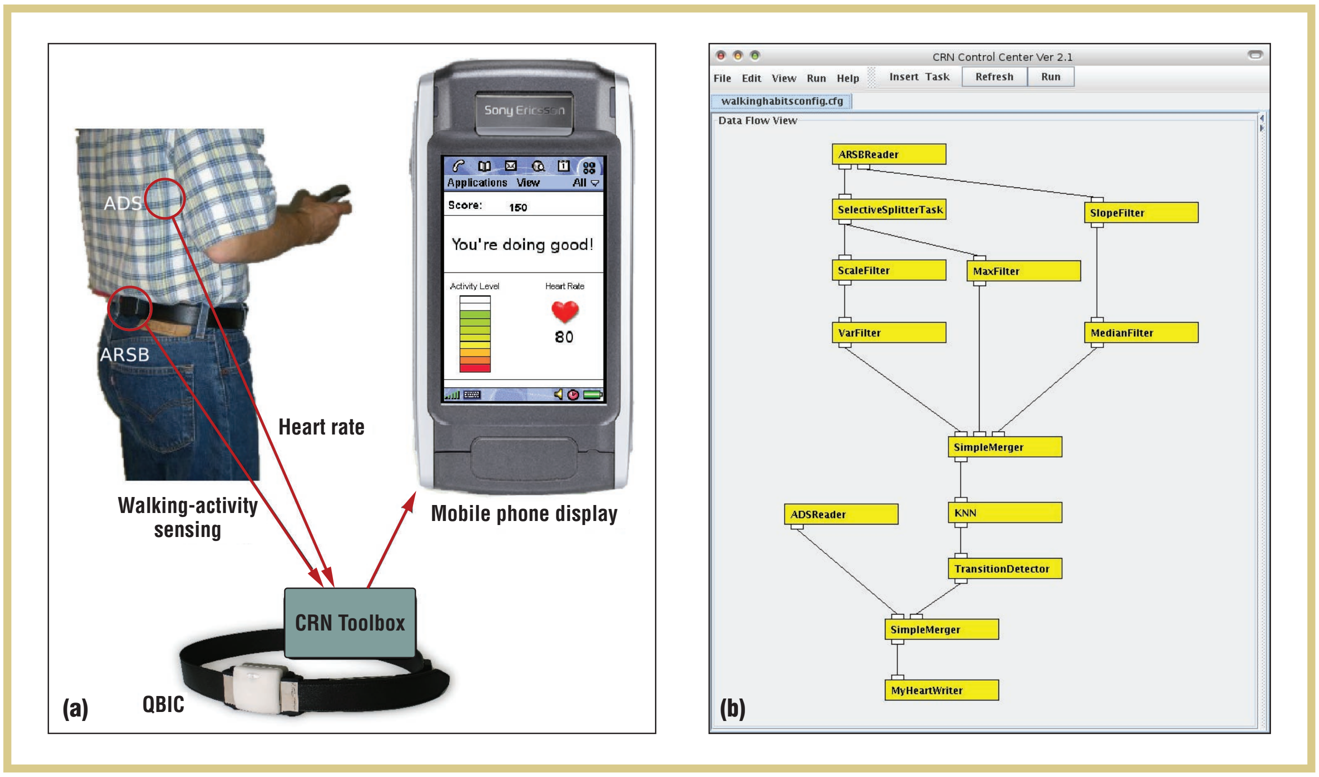


Figure 4. Walking-habits monitoring example: (a) phone visualization; (b) CRN Toolbox configuration.

figuration. This project required reader tasks to capture data from the belt sensor unit (ARSBReader) and heart rate belt (ADSReader). It also involves several filters, a classifier (KNN), and a writer to communicate to the mobile phone application (MyHeartWriter).

The visualizations on the phone showed activity level, heart rate, and recommendations based on detected activities. (In our ongoing work, we use further sensors at the limbs to capture diverse activities.)

Mixed-reality car-parking game

We designed a car-parking game to explore the use of wearable systems in computer games.[11] The game plot features the player helping a virtual driver fit a virtual car into a parking spot. The player does so by weaving hand and arm gestures while facing a virtual scene at the roadside, where a parking spot is available between other vehicles. Figure 5 shows a screenshot of the scene. The game simulates the driver's and car's behavior, which follow the gesture commands. The goal is to perform this guiding task as quickly and safely as possible—in particular, avoiding collisions with other cars and obstacles.

In this application, the CRN Toolbox recognizes gestures from the player's glove. Its task is to detect five gesture commands in the continuous data stream from the glove: forward, backward, turn left, turn right, and stop. We used acceleration and gyroscope sensors in three axes from an Xsens unit attached to the glove. The gesture-spotting procedure uses an explicit time series segmentation algorithm (SWAB), followed by a class-specific feature similarity search (Similarity). Next, ConfidenceMerger fuses the individual gestures. Then, StringMap maps the retrieved gestures to game commands, and TCPWriter transmits them to the game simulation and graphics engine.

We used the game as a demonstrator for tutorials and student courses. We built recognition models for 16 different gestures. Hence, every player could customize the system by selecting five gestures according to individual preferences. This customization simply involved exchanging configuration files for recognition tasks.

User evaluation

Right from its very first days, the CRN Toolbox has been a community project. The positive user feedback and the growing number of tasks indicate that our approach is well perceived. However, a thorough quantitative evaluation of middleware and programming tools such as the CRN Toolbox is difficult.[5] Although we haven't yet performed a controlled assessment, we do have some empirical (in some cases, quantitative) results that support our view of its usefulness.

Experience with students

As tables 2 through 4 show, we have widely used the Toolbox in student classes, and more than 60 students have worked with it.

A class of 19 fourth-semester com-

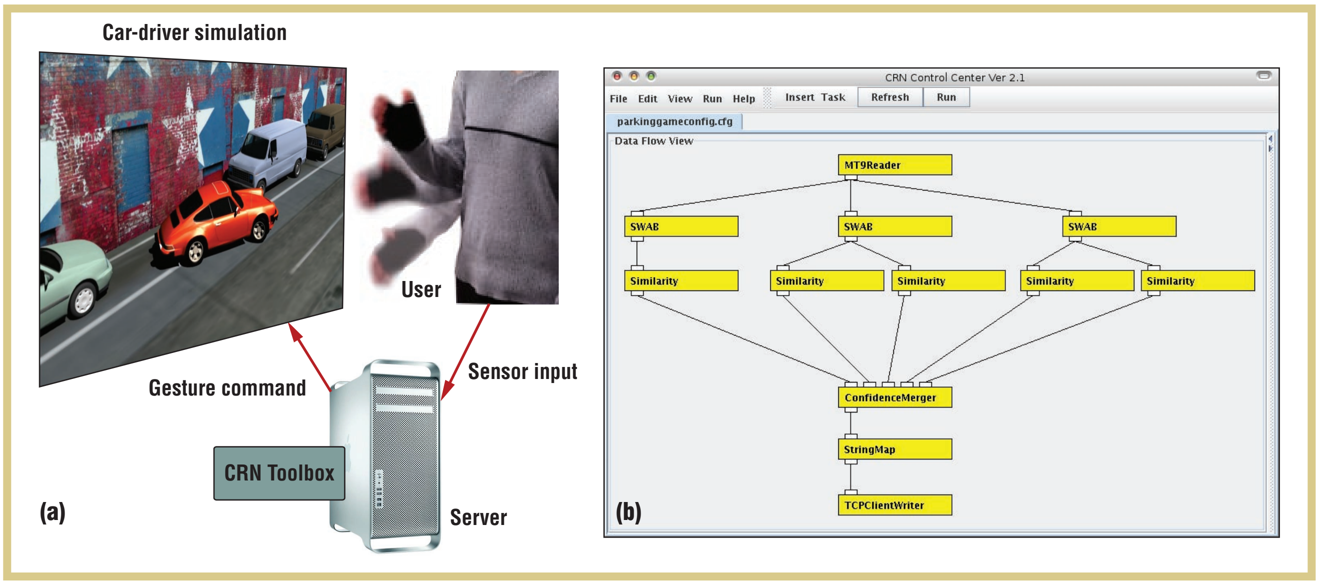


Figure 5. Car-parking game example: (a) a user manipulating a scene; (b) CRN Toolbox configuration.

puter science students implemented an application with the Toolbox to control the computer games Pong and Tetris by shaking a motion sensor in different directions. A typical solution for recognizing these gestures consists of five Toolbox components (approximately 1,200 LOC). They

- acquire data from sensors,
- apply filters (mean, variance),
- classify gestures using the KNN algorithm,
- send results via TCP, and
- manage the data flow.

The exercise also included the implementation of a new Toolbox task for reading from TCP sockets. The students were only Java beginners and had never programmed C++ before, yet with the Toolbox they all solved the recognition problem within 20 hours. Four of them also completed the Java game in that time.

Evaluation from researchers

We used the CRN Toolbox in an activity recognition tutorial at the 10th International Symposium on Wearable Computers (ISWC 2006). We asked 12 participants to rate their impressions after working with the Toolbox for four hours. Ten completed the tutorial feedback form. On average, they rated themselves as advanced software programmers with some knowledge of C++ but little experience in context recognition. They reported average durations of 10 minutes (maximum 30 minutes) to understand the four tasks of the tutorial, 15 minutes (maximum 30 minutes) to implement and run solutions with the Toolbox, and 20 minutes to debug their configuration, if needed.

We received many positive comments, such as "good and fast platform for application development," "one can click filters together," "easy to understand, easy to use," and "you don't have to reinvent the wheel." Criticism focused on missing documentation materials. Our current work addresses this issue by using automatic documentation tools and web platforms more intensively.

As a framework, the CRN Toolbox introduces some processing overhead. A prominent aspect in our design is the between-task communication, required in most useful configurations. This task relies on a common packet format to exchange all media types. Besides the payload, each packet contains a time stamp, a sequence number, and a payload pointer, totaling 16 bytes. In a typical scenario, such as the hospital support system discussed earlier, raw sensor-data packets have the highest transmission rate. In that example, an MT9Reader acquired a 9-channel Xsens MT9, requiring 36 bytes for one sample. In the default configuration, the reader outputs each sample as a separate data packet. This yields a total overhead of 44 percent. For packet rates above 100 Hz, such as in audio, the CRN Toolbox decreases the effective overhead by transferring multiple samples in one packet.

Most current applications of the Toolbox don't exploit its distributed-processing capabilities. We intend to use this feature in more complex applications in the near future. We also plan to investigate combining the CRN Toolbox with existing pervasive middleware frameworks that often rely on activity and context recognition services—precisely what the CRN Toolbox provides. ▣

ACKNOWLEDGMENTS

We express our gratitude to all the students and researchers who contributed to the development of the CRN Toolbox. This project is partly supported by the European Union WearIT@Work and MyHeart projects.

REFERENCES

1. P. Costa et al., "The Runes Middleware for Networked Embedded Systems and Its Application in a Disaster Management Scenario," *Proc. 5th IEEE Int'l Conf. Pervasive Computing and Communications* (PerCom 07), IEEE CS Press, 2007, pp. 69–78.

2. J. Hill et al., "System Architecture Directions for Networked Sensors," *ACM Sigplan Notices*, vol. 35, no. 11, 2000, pp. 93–104.

3. T. Weis et al., "Rapid Prototyping for Pervasive Applications," *IEEE Pervasive Computing*, vol. 6, no. 2, 2007, pp. 76–84.

4. S. Li et al., "Event Detection Using Data Service Middleware in Distributed Sensor Networks," *Telecommunication Systems*, vol. 26, nos. 2–4, 2004, pp. 351–368.

5. K. Edwards et al., "The Challenges of User-Centered Design and Evaluation for Middleware," *CHI Letters*, vol. 5, no. 1, pp. 297–304.

6. C. Becker et al., "PCOM: A Component System for Pervasive Computing," *Proc. 2nd IEEE Conf. Pervasive Computing and Communications* (PerCom 04), IEEE CS Press, 2004, pp. 67–76.

7. D. Crockford, *The Application/json Media Type for JavaScript Object Notation (JSON)*, IETF RFC 4627, July 2006; www.ietf.org/rfc/rfc4627.txt.

8. K. Adamer et al., "Developing a Wearable Assistant for Hospital Ward Rounds: An Experience Report," to be published in *Proc. Int'l Conf. Internet of Things* (IOT 08), Springer, 2008; www.the-internet-of-things.org.

9. D. Bannach et al., "Distributed Modular Toolbox for Multimodal Context Recognition," *Proc. 19th Int'l Conf. Architecture of Computing Systems*, LNCS 3894, Springer, 2006, pp. 99–113.

10. T. Stiefmeier et al., "Event-Based Activity Tracking in Work Environments," *Proc. 3rd Int'l Forum Applied Wearable Computing* (Ifawc 06), TZI Universität Bremen, 2006; http://spring.bologna.enea.it/ifawc/2006/proceedings/IFAWC2006_10.pdf.

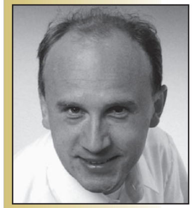
11. D. Bannach et al., "Waving Real Hand Gestures Recorded by Wearable Motion Sensors to a Virtual Car and Driver in a Mixed-Reality Parking Game," *Proc. IEEE Symp. Computational Intelligence and Games* (CIG 07), IEEE Press, 2007, pp. 32–39.

12. O. Amft, M. Kusserow, and G. Tröster, "Probabilistic Parsing of Dietary Activity Events," *Proc. 4th Int'l Workshop Wearable and Implantable Body Sensor Networks* (BSN 07), Ifmbe 13, Springer, 2007, pp. 242–247.

the AUTHORS

David Bannach is a doctoral candidate and a member of the research staff at the Embedded Systems Laboratory of the University of Passau. His research interests focus on software systems for context-aware computing. He received his diploma in computer science from ETH Zurich. Contact him at ESL, Univ. of Passau, Innstrasse 43, 94032 Passau, Germany; david.bannach@uni-passau.de.

Oliver Amft is a doctoral candidate in the Wearable Computing Lab at ETH Zurich. His research interests focus on pervasive healthcare and personal-assistant systems, including embedded systems, pervasive sensing, and pattern recognition for physiology, activity, and behavior awareness. He received his MSc in electrical engineering from Chemnitz Technical University. He is a member of the IEEE. Contact him at Wearable Computing Lab., ETH Zurich, c/o Electronics Laboratory, Gloriastrasse 35, CH-8092 Zurich; amft@ife.ee.ethz.ch.



Paul Lukowicz is a full professor and chair of Embedded Systems and Pervasive Computing at the University of Passau. His research interests include wearable and mobile computer architecture, context and activity recognition, high-performance computing, and optoelectronic interconnection technology. He received his PhD in computer science from the University of Karlsruhe, Germany. Contact him at ESL, Univ. of Passau, Innstrasse 43, 94032 Passau, Germany; paul.lukowicz@uni-passau.de.