Waving Real Hand Gestures Recorded by Wearable Motion Sensors to a Virtual Car and Driver in a Mixed-Reality Parking Game

David Bannach,[1] Oliver Amft,[2] Kai S. Kunze,[1] Ernst A. Heinz,[3] Gerhard Tröster,[2] Paul Lukowicz[1, 3]

[1] Embedded Systems Lab, University of Passau, Passau
{david.bannach, kai.kunze, paul.lukowicz} @ uni-passau.de

[2] Wearable Computing Lab, ETH Zürich, Zürich
{amft, troester} @ ife.ee.ethz.ch

[3] Computer Systems and Networks (CSN), UMIT, Hall in Tyrol
{ernst.heinz, paul.lukowicz} @ umit.at

Abstract— **We envision to add context awareness and ambient intelligence to edutainment and computer gaming applications in general. This requires mixed-reality setups and ever-higher levels of immersive human-computer interaction. Here, we focus on the automatic recognition of natural human hand gestures recorded by inexpensive, wearable motion sensors. To study the feasibility of our approach, we chose an educational parking game with 3-D graphics that employs motion sensors and hand gestures as its sole game controls. Our implementation prototype is based on Java-3D for the graphics display and on our own CRN Toolbox for sensor integration. It shows very promising results in practice regarding game appeal, player satisfaction, extensibility, ease of interfacing to the sensors, and – last but not least – sufficient accuracy of the real-time gesture recognition to allow for smooth game control. An initial quantitative performance evaluation confirms these notions and provides further support for our setup.**

Keywords: Game Control, Gesture Recognition, Immersive Human-Computer Interaction, Java-3D, Mixed Reality, Motion Sensors, Wearable Computing

I. Introduction

A central aspect of progress for edutainment and computer gaming applications is their steady increase in the immersiveness of the user experience. Historically, this routinely sprang from ever more involved and complex plots as well as crisper and more spectacular 3-D graphics. Then, games combined the two resulting in blazing graphics on top of challenging interactive stories and other such things of personalization. Game control, however, remained quite crude, mostly relying on text input, mouse clicks, joypads, and joysticks. Maybe this is why certain genres supporting much nicer control devices flourish so well. Car racing games, for instance, achieve an extremely immersive user experience by means of realistic steering wheels and gear shifts paired with real-time force feedback as their preferred means of game control.

During roughly the past 5 years, a general shift towards better and more immersive game control started to take hold. Pioneered by the "EyeToy" and dancing mats for Sony game consoles, not only academics but also the entertainment industry realized that people's real-world physical actions need to directly affect their playing reality. Contrasting with others, we envision to employ inexpensive wearable sensors to achieve this – preferably tiny motion sensors worn by people on their bodies (e.g., integrated into their clothes and other personal accessories like watches or jewelry). Such body-mounted sensors provide an inexpensive basis for motion analysis while letting users move and roam about freely, independent of any additional infrastructure. Hence, we deem them superior to and prefer them over other approaches pursuing similar goals. Current hardware developments seem to support our notion in this respect. Select gaming consoles (e.g., Nintendo DS and Wii) and other electronic gadgets already come equipped with integrated motion sensors.

This adds immediate relevance to the work presented here. Our real-time parking game lets players direct a virtual car and driver waving common, natural hand gestures in the real world. Tiny motion sensors, unobtrusively mounted in gloves, record the movements and send them via a wireless connection to the remote computer running the main application with 3-D graphics display. There, automatic gesture analysis and recognition takes place before initiating the virtual actions directed by the player's real gestures.

After discussing related work in Section II, we focus on the game and our prototype implementation based on our own CRN Toolbox and Java-3D in Section III. Then, we elaborate on the intricacies of our motion-sensed gesture recognition in Section IV and present an initial performance evaluation for the recognition scheme in Section V. Finally, we conclude with some further discussions and thoughts on future work.

II. Related Work

Over the years, countless new add-ons for more immersive human-computer interaction in gaming applications were proposed. Most of them never even made it beyond paper design. The rest, however, are still numerous enough to feed a surprisingly large pool of commercially available such gadgets: steering wheels and gear shifts, infrared and laser guns with according armor, infrared steering helmets, touch- and pressure-sensitive mats for step-based activities like dancing, cheap head-mounted displays, and video overlaying made popular by the "Eyetoy" of Sony to name but a few. Compared with our approach based on tiny and inexpensive wearable sensors, the above mentioned gaming gadgets all feature disadvantages of clumsiness, relatively high costs, and lack of flexibility among others.

By now, many independent researchers have demonstrated the suitability and excellent further potential of body-worn sensors for automatic context and activity recognition, e.g., [1], [2], [3], [4], [5], [6], [7], [8]. The available scientific literature reports about successful applications of such sensors to various types of activities, ranging from the analysis of simple modes of locomotion [3] to more complex tasks of everyday life [2] and even workshop assembly [9].

There are much fewer publications, however, about using wearable sensors in gaming applications. Several co-authors of this text reported on an initial experiment with wearable motion sensors for real-time recognition tasks in games of martial arts [10]. The multiplayer game "Collective Calm" [11] relies on gloves with integrated galvanic skin-response sensors to determine the calmest of all participants as the final winner. In [12], wearable pressure sensors integrated into body protectors help to control and decide the counting of points for Taekwando. Supposedly helping children with their Kung Fu education, [13] introduces some kind of interactive computerized toy ball.

Instead of relying on wearable sensors, various systems employing image- and video-processing techniques pursue similar goals of immersive human-computer interaction as we do. Noteworthy in this context are [14], [15], [16], [17] to name but a few. On top of relatively high system costs, these works suffer from the usual drawbacks of video-based approaches, i.e., high sensitivity for lighting conditions and demanding requirements on equipment and infrastructure.

III. The Game and Our Implementation Prototype

Fig. 1. Screen display of the parking scene with the game in progress

The plot of our game features the player helping a virtual driver to fit the latter's virtual car into a parking lot. The player does so by making real gestures with his arms and hands while facing a virtual scene at the roadside where a parking spot is available between other vehicles.

The game play and scene evolve as follows. A car on the road waits to get in. Waving some common natural gestures, the player directs the virtual driver's simulated steering. Thus, he effectively guides the waiting car into the parking space. The goal is to perform this guiding task as fast and safely as possible, in particular avoiding collisions with other cars and obstacles (see Fig. 1 for a sample screen display of the game in action).

The remainder of this section elaborates on the design, structure, and other implementation details of our application prototype for playing the parking game. We aimed at a rapid solution built from reusable components. Hence, the challenge was to avoid over-specialized optimizations while at the same time maintaining instantaneous game reactions and entertainment value. Furthermore, the design was chosen with teaching purposes in context awareness and activity / gesture recognition in mind. The implementation comprises the following parts: (1) the CRN Toolbox as middle-ware for signal processing and pattern recognition, (2) the gesture detection and sensor interface, and (3) the driver simulation and car movement.

A. The CRN Toolbox

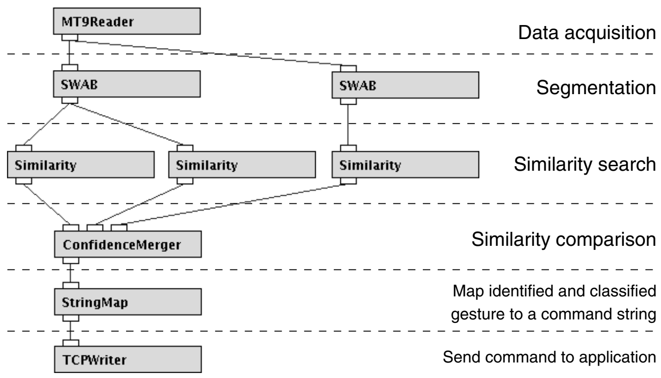


Fig. 2. Simplified view of toolbox configuration for the parking game

We consider the task of recognizing gestures as a specialization of context recognition in general where data from sensors mounted on the body and in the environment are captured continuously and processed in an on-line fashion. To facilitate the creation of new application prototypes in such fields, we developed the Context Recognition Network (CRN) Toolbox [21]. This software toolbox allows to quickly build distributed, multi-modal context recognition systems by simply plugging together reusable, parameterizable components. Thus, the toolbox simplifies the steps from prototypes to final implementations that might have to fulfill real-time constraints on low-power mobile devices. Moreover, it facilitates portability between platforms and fosters easy adaptation and extensibility. The toolbox also provides a set of ready-to-use parameterizable algorithms including different filters, feature computations and classifiers, a run-time environment that supports complex synchronous and asynchronous data flows, encapsulation of hardware-specific aspects including sensors and data types (e.g., `int` vs. `float`), and the ability to outsource parts of the computation to remote devices.

Toolbox components are tasks executed in parallel. They continuously process data received at the *in-port* and put the results to the *out-port*. Connections can be defined between out-ports and in-ports. Special *Reader*-tasks acquire the data from external sources like sensors, TCP-streams, or files. They have no in-port. Similarly, *Writer*-tasks write data to external devices. For the gesture recognition of the parking game, we used the SWAB and similarity-search components discussed in Section IV. A simplified version of the toolbox configuration is displayed in Fig. 2, showing the sensor-specific *MT9Reader* task as source of the data flow. Several differently parametrized instances of the SWAB and similarity-search tasks process the data streams until the final *TCPWriter* writes the recognition results to a TCP-stream.

B. Overall Implementation Structure

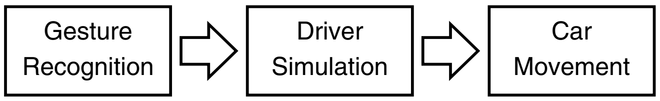


Fig. 3. Structural components of our implementation prototype

As summarized in Fig. 3, our implementation prototype consists of three main components that we describe below.

1) Gesture Recognition: The player is wearing a motion sensor on the right hand. The data from this motion sensor is continuously sampled and processed by our context recognition toolbox for recognizing gestures of the player. When a gesture is recognized by the toolbox, it sends the corresponding command to the driver simulation.

2) Driver Simulation: The driver simulation receives commands from the gesture recognition component and accordingly controls the speed and steering angle of the virtual car. It simulates the driver's reaction on perceived gestures. In contrast to the command which is instantaneous, the simulated reaction has an extent in time space. Overlap of reactions is handled by the driver simulation.

3) Car Movement: The car movement component is ensuring the correct movement of the car in the virtual scene. It is updating the position and orientation of the car according to the current physical properties of the car and the restrictions of the scene (e.g., collisions).

C. Sensor Interface and Data Recording

For the hand gesture interface a standard sports glove was used with the motion sensor[1] attached on the back of the hand. The motion sensor provides calibrated 3D acceleration, gyroscope and magnetic field (compass) sensor data, however only acceleration and gyroscope data was used in this work. The sensor data was recorded at 100 Hz by the CRN Toolbox MT9Reader task and forwarded to the gesture detection tasks within the toolbox.

In order to obtain a simple gaming system we restricted the gestures to the user's right hand. A total of 16 different gesture classes, potentially useful to navigate a car remotely, were defined. Although the sensor was attached to the right hand the users supported some gesture classes with their left hand to achieve symmetry or a more natural motion execution.

[1]Model: MT9, XSens BV, NL

D. Driver simulation and car movement

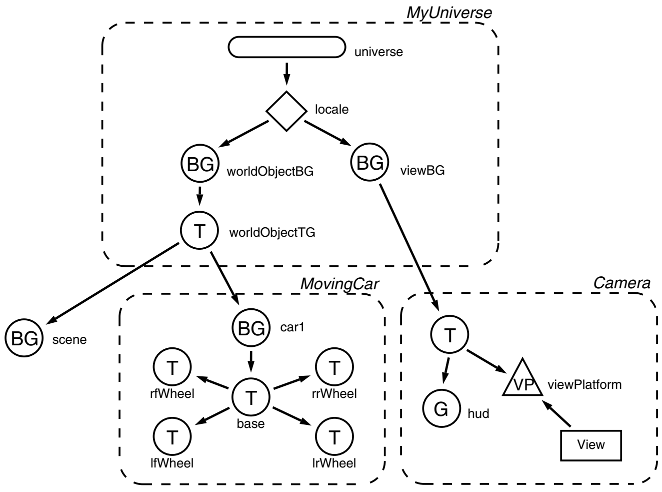


Fig. 4. The (simplified) Java3D scene graph.

We found the Java3D scene graph suitable for our purpose as it provides means for interaction and animation while keeping the complexity at a low level. We also found a way to use the 3D-modeling tool Blender[2] for creating the 3D scene and then loading it into the scene graph.

Driver simulation, car movement, and visualization are realized using the scene graph depicted in Figure 4. The class *MyUniverse* provides the base of the scene graph where all objects that show up in the scene or that model the behavior of scene objects are attached. The *Camera* class covers the details of how the scene is presented to the user. The *MovingCar* class aggregates car model, driver simulation, and car movement. Separate transform groups (T) for wheels and car body allow the animation of car movement and wheel rotation. Figure 5 shows the classes that affect the behavior of the moving car. The behaviors are divided in those for driver simulation and those for car movement. The *ControlPort* receives commands from the gesture recognition component via a TCP socket and triggers the *CommandBehavior*. The *CommandBehavior* decides how to react on each command and sends new target values for speed and steering angle to the *SpeedInterpolator*. The *SpeedInterpolator* animates the actual reaction on a command and handles the overlap in time space where the most recent reaction has the highest priority. The speed will smoothly be animated to zero when no command is received for a definable period of time. The *CarMovementBehavior* class is triggered with each rendered frame. It updates the position and orientation according to the current state of the car and the time passed since the last rendered frame. The state includes static properties like the physical configuration of the car as well as dynamic properties like speed, steering angle, and collision state. The latter is controlled by the *CollisionBehavior* which is detecting intersecting objects. For performance reasons, we

[2]Open source 3D graphics tool: http://www.blender.org

use simplified versions of the rendered models for collision detection.

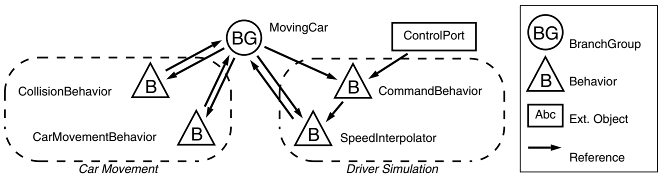


Fig. 5. Behaviors for driver simulation and car movement controlling the animation of the virtual car.

We used Blender to readily compose the virtual scene and we enriched it with freely available 3D models. Blender can export the complete scene including models, materials, textures, lights, and cameras to the X3D[3] standard. Furthermore, there is a loader component available for Java3D that is capable of loading X3D encoded scenes into the scene graph. Many other loaders either did not work properly or only partially loaded the models. We used the X3D loader in our application for models, materials, textures, camera, and lights.

IV. Motion-Sensed Hand Gesture Recognition

A. Pre-Recording and Pre-Selection of Gesture Classes

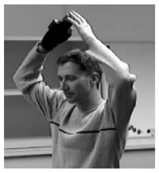
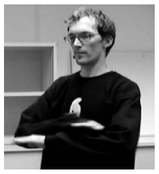
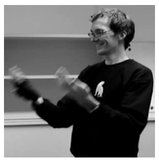
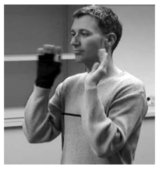
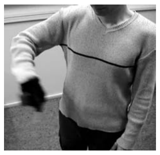
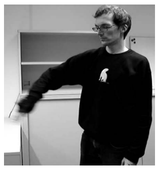
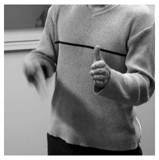
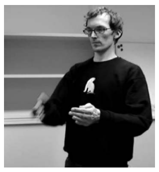
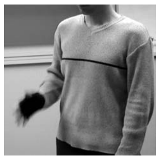
To estimate the detection algorithm parameters and evaluate the detection performance, gesture data was recorded in advance. For each class 20 to 40 individual instances were acquired from two user resulting in a total of 1768 gesture instances in 16 classes. From this initial set five classes were selected to control the game. However, every player may configure a personal set of gestures for controlling the game. A brief description including a picture of the gesture execution is presented in Tab. I.

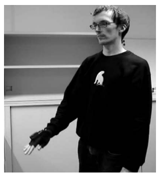
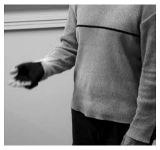
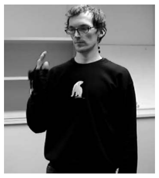
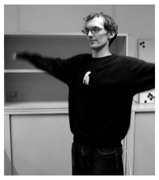
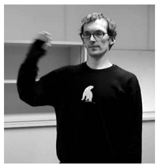
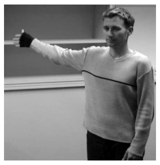
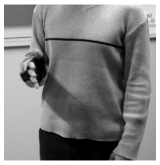
The gesture classes included periodic, e.g. circular motion of the hand, as well as non-periodic motions, e.g. pointing with the hand. For periodic gestures begin and end of the motion are defined arbitrary during execution in contrast to non-periodic gestures, that can be described best as motion event with a clear start end end point. Begin and end of the motions were defined by annotating the recorded data in a post-processing step. For periodic motions, the annotation bounds were chosen arbitrary for each class, for non-periodic motions the bound were defined to cover motion onset, e.g. moving the hand up, and the rather static middle phase, e.g. the actual pointing during a hand pointing gesture. The return motion towards the hand rest position was excluded. This is necessary since a natural driver would already spot the gesture during the motion and not just after the hand returned to the rest position. While being more natural, this approach may lead to less distinctive gesture classes for the algorithm since some cues from the hand motion are missing.

[3]Extensible 3D (X3D), ISO/IEC 19775:2004, http://www.web3d.org

TABLE I

Description of the considered gesture classes.

Class name	**Gesture conduction**	**Gesture description**
Overhead		Upward movement of the arm, holding hand over head, supported by left hand, non-periodic.
Stop		Opposed movement of the lower arms bend at ∼90 degrees from upper arms, starting by moving towards each other and return, hands in pronation, non-periodic. Default gesture for "Stop" command in the game.
To me		Movement of the lower arms from extended position towards the chest, return and repeat, hands in supination, periodic. Default gesture for "Driving forward" command in the game.
Away		Movement of the lower arms from near chest position towards extended position, return and repeat, hands in pronation, periodic. Default gesture for "Driving backward" command in the game.
Turn left		Left rotation movement of the right arm, lower arm and hand pointing down, periodic. Default gesture for "Turn left" command in the game.
Turn right		Right rotation movement of the right arm, lower arm and hand pointing down, periodic. Default gesture for "Turn right" command in the game.
Closer left		Left rotation of the lower right arm, held in ∼90 degrees of the upper arm, hand in neutral, periodic. Left hand held motionless was used to display an obstacle.
Closer right		Right rotation of the lower right arm, held in ∼90 degrees of the upper arm, hand in neutral, periodic. Left hand held motionless was used to display an obstacle.
Sharp stop		Single fast up-down movement of the lower right arm, hand in neutral, non-periodic.

Class name	Gesture conduction	Gesture description
Slow down		Slow up-down movement of the lower arms at ∼90 degrees of the upper arm, hand in pronation, periodic.
Arm up		Slow up movement of the lower right arm at ∼110..90 degrees of the upper arm, hand in supination, periodic.
Chop		Fast up-down movement of the lower right arm at ∼80..110 degrees of the upper arm, hand in neutral, non-periodic.
Jump jack		Up-down movement of the extended arms, hand in pronation, non-periodic.
Wave		Waving motion of the lower right arm with hand at height of head, periodic.
Point		Pointing with extended arm in line and height of shoulder, hand in neutral, non-periodic.
Sideways		Inward-outward movement of the lower right arm at ∼90 degrees of the upper arm, hand in neutral, periodic.

B. Gesture Detection in Detail

Our system has to deal not only with the gesture classification but also with the problem of gesture spotting. Thus the control gestures need to be extracted from a continuous data stream in which they are mixed with other random motions (e.g. user scratching the head). Those other motions are referred to as the 'NULL class'. The difficultly of extracting the relevant gesture stems from the fact that human arms motions are very rich and subject of few constraints. As a consequence, it is not possible to reliably model the NULL class. This is in sharp contrast to, for example speech recognition, where the NULL class consists essentially of silence and some very seldom occurring sounds such as coughing. Another issue are variations in the length of relevant gestures. This means that a simple fixed sliding window search method will not be able to identify all gestures. In a brute force approach, sensor data may be scanned for relevant gestures by considering each sample as potential gesture starting point and each following sample as possible end. However, considering practical data rates of 20 to 100 Hz this approach is obviously not practicable for real time recognition, even by constraining the search bounds.

The vast majority of related work in the area of gesture recognition sidesteps the problems by defining specific, simple to identify and often fixed length gesture start/stop signals. As pointed out above, our aim is to keep interaction as natural as possible so that such fixed start/stop markers are not feasible.

In this paper we adapt and extend previous gesture spotting work by our group ([18], [19]). Our method is based on three main ideas: (1) using online piecewise linear approximation segmentation of signal to reduce the computational complexity of the search, (2) performing the search on a data adapted, dynamically changing window size, and (3) fusing classifiers with statistically largely independent sources of errors to filter out false positives.

1) Data Segmentation: The procedure presented in this work applies an online segmentation step to reduce the number of search points drastically. The algorithm utilized for this step is Sliding window and bottom-up (SWAB) [20]. SWAB works by moving a segment buffer along the data, linearly returning segments and continuously adding new data. The segmentation is obtained by testing the approximation of the data signal by linear regression lines. A detailed description can be found in Keogh et al. [20]. For the gestures used in this work raw acceleration and gyroscope sensor axes were used.

2) Feature Similarity Search: In the second step the segmentation points are used to search for potential gestures using a feature similarity detection algorithm. The search is performed by comparing features of a data section under investigation to a trained pattern. For a given segmentation point, the history of sensor data is analyzed from a lower up to an upper search bound. These bounds are determined in the training step from minimum/maximum overlaps of the annotated events and the segmentation points. The feature comparison is achieved by computing the Euclidean distance between the features of the data section under investigation during the search and the trained pattern. A distance threshold, obtained during training, is used to omit unlikely sections. The advantage of this algorithm is that it works as a one class classifier. It can be applied to detect one relevant class and it does not make any further assumption about the data not included in the relevant class (NULL class). Multiple instances of this feature similarity search can be used to spot different classes independently, admitting an independent feature selection for each class. Finally the obtained distances are converted to confidences by normalization using the distance threshold.

3) Similarity Merging: In the third and final step, all independent gesture detection results of the feature similarity searches are merged. The algorithm used here is based on a sliding buffer of gesture sections. Sections are entering from the similarity search tasks. For each entering section the collision (overlapping of the gesture section with sections already in the buffer) are resolved in favor for the highest confidence section. Sections are released from the buffer after a timeout. We refer to this merging algorithm similarity comparison. This step in the procedure works effectively as a classifier, returning the most likely gesture section from the independent search tasks.

This gesture detection and classification approach can be customized to integrate very different gesture types. For the gestures classes used in our work different SWAB segmentations were utilized and individual feature sets were chosen for each class. Furthermore the search bounds and distance thresholds are adapted for each similarity search during training. For the intended online gaming system delays must be kept low. This was achieved by using relatively short gestures and minimizing the buffers for the SWAB and similarity comparison algorithms at the expense of a reduced detection precision.

4) Integration with the CRN Toolbox: The three algorithms for segmentation, feature similarity search and the similarity comparison have been integrated as individual tasks in the CRN Toolbox. Fig. 2 shows a data flow chart of the individual detection tasks. For the ParkingMaster game the tasks are used to spot relevant gestures online in the continuous sensor data. The modularization of the detection procedure permits fast reconfiguration for a certain set of gestures. This can be used for personalization of the gaming system. As detailed below, training data for all gestures was acquired to determine the algorithm parameters in advance.

V. Quantitative Performance Evaluation

In order to evaluate the game, we analyzed the gesture detection performance. Training and testing was performed based on the previously recorded data sets. To account for variations in the data set a 4-fold cross-validation procedure was used to determine training and testing data set for the detection procedure. For training 3 of 4 parts of the data were used. Evaluation was performed on the left out data part. This procedure was repeated until all 4 parts were used for testing once. The boundaries of the parts were adapted to avoid intersecting relevant gesture data sections.

To analyze the detection performance, we utilized the metrics *Precision* and *Recall* commonly used for evaluation in Information Retrieval. These metrics are derived as follows:

$$Recall = \frac{\text{Recognized gestures}}{\text{Relevant gestures}} \qquad (1)$$

$$Precision = \frac{\text{Recognized gestures}}{\text{Retrieved gestures}} \qquad (2)$$

Relevant gestures corresponds to the manually annotated number of actually conducted gesture instances in a class. Retrieved gestures represents the number of gestures returned by the algorithm. Finally, recognized gestures refers to the correctly returned number of gestures. Higher values indicate better performance for both metrics.

Tab. II presents a precision-recall evaluation for all 16 classes for single-user data. The evaluation is based on the detection results returned by the similarity comparison algorithm. Since single-user data was used the detection performance corresponds to the result that can be achieved when the system is trained for the user. It can be seen, that the detection performs well for most of the classes: scores greater than 0.8 can be achieved for both, precision and recall. A small group of gestures (“Away”, “Closer left, “Sharp stop”, “Point”) catch more insertions or miss events (deletions) compared to the rest.

While the results for all 16 classes provide an overview on the detection performance for each of the considered gesture types, only five classes are used within the game. We selected a default class configuration based on good detection performance and intuitive applicability in the gaming scenario. Fig. 6 shows the individual performance of the selected classes. The result is based on data from two users amounting to a total of 751 gesture instances contained in the five classes. The results differ slightly from the evaluation of all 16 classes since in this configuration, the similarity comparison step is connected to five similarity search tasks only. In this configuration a high recall is achieved for all gestures, however at the expense of a low precision for the gesture “Turn left”.

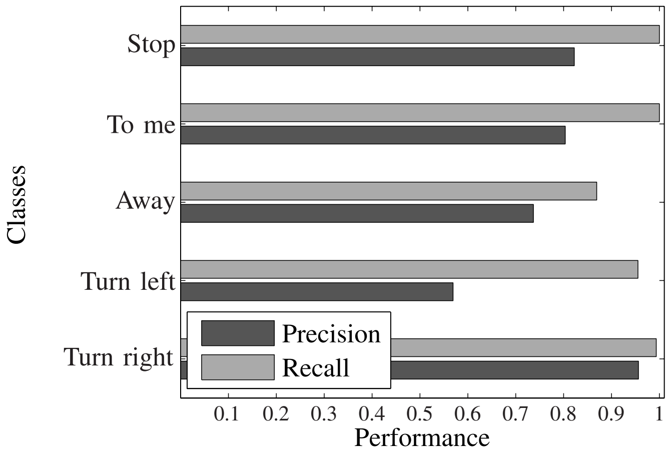


Fig. 6. Precision-recall chart for the single-user evaluation of the default gesture classes selected for the game. Best performance is found towards high precision and recall.

Furthermore we evaluated the detection performance for a new user of the game. This test corresponds to the hardest scenario where the system must cope with gestures from a not previously seen user. The evaluation was performed by training the feature similarity algorithm with the gestures from the first user and testing on the second. The procedure was subsequently repeated by training on the second user. Fig. 7 presents a performance comparison for the unseen-user and single-user evaluations using the five default gestures.

TABLE II

SUMMARY FOR THE SINGLE-USER DETECTION PERFORMANCE FOR ALL CLASSES.

Metric	Classes															
	Overhead	Stop	To me	Away	Turn left	Turn right	Closer left	Closer right	Sharp stop	Slow down	Arm up	Chop	Jump jack	Wave	Point	Sideways
Relevant	33	37	147	145	268	154	175	63	129	103	87	46	68	47	148	118
Retrieved	41	38	148	119	274	156	260	58	152	103	75	38	57	48	125	129
Recognized	33	34	143	119	256	153	173	58	124	103	75	38	57	47	117	115
Deletions	0	3	4	26	12	1	2	5	5	0	12	8	11	0	31	3
Insertions	8	4	5	0	18	3	87	0	28	0	0	0	0	1	8	14
Recall	1.00	0.92	0.97	0.82	0.96	0.99	0.99	0.92	0.96	1.00	0.86	0.83	0.84	1.00	0.79	0.97
Precision	0.80	0.89	0.97	1.00	0.93	0.98	0.67	1.00	0.82	1.00	1.00	1.00	1.00	0.98	0.94	0.89

A sweep using different confidence thresholds was used to create the precision-recall curves. The result clearly indicates that a good performance is achieved for a user known to the system by previous training. However, the system returns a higher error rate for an entirely new user.

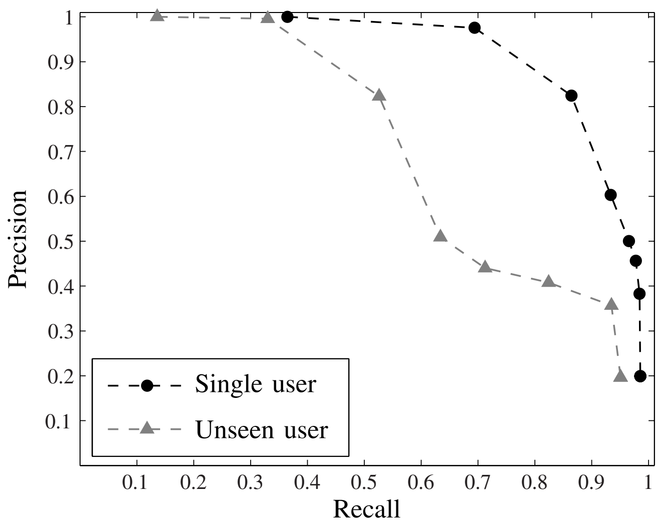


Fig. 7. Precision-recall comparison (confidence threshold sweep) of the single-user and unseen-user evaluations. This analysis is based on the default game gestures. Best performance is found towards the top-right corner (high precision, high recall).

VI. DISCUSSION, CONCLUSION, AND FUTURE WORK

A. Evolving the Game and Our Implementation Prototype

Thanks to its unique user interface and despite its conceptual simplicity, our parking game seems to enjoy broad appeal. Nearly all test players (including ourselves!) report the game to be entertaining and fun to play. Manifold extensions of play are possible, of course, and straightforward to imagine: add other kinds of vehicles to park in, exact timing, potential damage effects with audio-visual feedback, different levels of ambient traffic on the road, and completely new scenarios (e.g., controlling a virtual traffic cop).

Moreover, the game is quite educational by nature because it requires real discipline and correct gesturing. Hence, we might consider slight variations of our gaming setup as a potential edutainment tool for learning how to gesture correctly. Taking this idea a step further, the game and its unique user interface might actually be interesting as a tool for therapy of certain injuries and/or impairments.

As for implementation prototyping, the CRN Toolbox proved its value. First and foremost, it hides the details of how to manage the distributed gaming application overall and how to interface to the sensors in particular. In addition, the toolbox also provides other kinds of middle-ware functionality and a broad library of standard algorithms for signal and data pattern processing. Last but not least, the CRN Toolbox offers communication interfaces to different programming languages (such as Java, C/C++, and MatLab) on heterogeneous platforms. Thus, it is well suited for rapid prototyping of demanding applications like the ones that we envision for the realm of computer gaming.

B. Recognition of Motion-Sensed Gestures

We have presented a procedure to spot gestures online in continuous data for controlling a 3D game. We subsequently evaluated the recognition performance for two typical configuration scenarios: single-user and unseen-user. The system was able to detect the 16 different gesture classes from simple motion sensors attached to a glove at one hand. The gesture classes included periodic and non-periodic movements. Since non-periodic gestures have different properties than periodic, e.g. signals for movement begin and end, as well as movement duration, they are difficult to detect with one algorithm. However both performed well for the procedure and scenario presented in this work. Due to the modular concept the system can easily be extended to the second hand or the whole body for other activity-based games.

The gesture spotting results indicate that the system works very well in personally trained configurations (single-user). As expected, the unseen-user evaluation indicated a drop in detection performance. However in practical gaming the system worked well. Here the performance difference was subjectively less visible. This difference may be due to the direct visual feedback in the gaming scenario whereas data for this evaluation was gathered in advance without the running game. We plan to evaluate the system with further users and analyze the variation of the detection results, e.g. the influence of force and energy related to the gaming situation. We expect that future game interfaces may incorporate such information to adapt the detection or give feedback to the user.

C. Usefulness of the CRN Toolbox

We used the parking game as a hands-on example for the activity recognition tutorial at ISWC 2006, the Intl. Symp. on Wearable Comp. (see `http://www.ife.ee.ethz.ch/~oam/projects/iswc2006/` for details). The ten participants of the tutorial (re-) implemented the gesture recognition and interfaced it with the rest of the game on their own. All participants enjoyed substantial experience in software engineering and programming, yet most were quite new to the field of context recognition (8 out of 10). The participants' main complaints related to the lack of good documentation for the CRN Toolbox. Other concerns pointed towards a still somewhat clumsy syntax of the configuration language and the need for a graphical user interface (GUI) to better grasp and visualize the notion of connections. Both these points are already remedied by now. On the positive side, the participants liked the encapsulation of the toolbox and found its concepts easy to understand even with only minimal documentation.

We also employed the CRN Toolbox in practical classes and three programming assignments at the University of Passau in 2006. Each assignment spanned three days with class times of 4h per day. Participants numbered 20 to 25 undergraduate students in their second year of study. Due to the students' lack of experience with programming in general and context awareness in particular, we picked only "toy" activity recognition tasks as exercises. Based on our favorite wearable motion sensors, the assignments included cursor control by trivial gestures, input of natural numbers, and recognition of walking as opposed to sitting. Surprisingly, several of the participants tackling cursor control managed to extend their assignment towards gaming. They actually implemented a working control to play simple classic games like Tetris and Pong by means of real hand movements.

REFERENCES

[1] O. Cakmakci, J. Coutaz, K. V. Laerhoven, and H.-W. Gellersen, "Context awareness in systems with limited resources." [Online]. Available: citeseer.nj.nec.com/cakmakci02context.html

[2] L. Bao and S. Intille, "Activity recognition from user-annotated acceleration data," in *Pervasive Computing*, F. Mattern, Ed., 2004.

[3] L. Seon-Woo and K. Mase, "Recognition of walking behaviors for pedestrian navigation," in *Proc. of the 2001 IEEE International Conference on Control Applications (CCA'01)*, 2001, pp. 1152–1155.

[4] J. Mantyjarvi, J. Himberg, and T. Seppanen, "Recognizing human motion with multiple acceleration sensors," in *2001 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 3494, 2001, pp. 747–752.

[5] E. A. Heinz, K. S. Kunze, S. Sulistyo, H. Junker, P. Lukowicz, and G. Tröster, "Experimental evaluation of variations in primary features used for accelerometric context recognition," in *Proceedings of the 1st European Symposium on Ambient Intelligence (EUSAI 2003)*, E. Aarts, R. Collier, E. van Loenen, B. de Ruyter (eds.), LNCS 2875, Springer-Verlag, 2003, pp. 252–263, ISBN 3-540-20418-0.

[6] N. Kern, B. Schiele, H. Junker, P. Lukowicz, and G. Tr̈oster, "Wearable sensing to annotate meeting recordings," in *Proceedings Sixth International Symposium on Wearable Computers ISWC 2002*, 2002.

[7] N. Kern, B. Schiele, and A. Schmidt, "Multi-sensor activity context detection for wearable computing," in *Proceedings of the 1st European Symposium on Ambient Intelligence (EUSAI 2003)*, E. Aarts, R. Collier, E. van Loenen, B. de Ruyter (eds.), LNCS 2875, Springer-Verlag, 2003, ISBN 3-540-20418-0.

[8] P. H. Veltink, H. B. J. Bussmann, W. de Vries, W. L. J. Martens, and R. C. van Lummel, "Detection of static and dynamic activities using uniaxial accelerometers," *IEEE Transactions on Rehabilitation Engineering*. Vol. 4, No. 4, pp. 375–385, 1996.

[9] P. Lukowicz, J. Ward, H. Junker, M. Staeger, G. Tröster, A. Atrash, and T. Starner, "Recognizing workshop activity using body worn microphones and accelerometers," in *Pervasive Computing*, 2004.

[10] E. A. Heinz, K. S. Kunze, M. Gruber, D. Bannach, and P. Lukowicz, "Using wearable sensors for real-time recognition tasks in games of martial arts – An initial experiment," in *Proceedings of the 2nd IEEE Symposium on Computational Intelligence and Games (CIG 2006)*, S.J. Louis, G. Kendall (eds.), pp. 98–102, IEEE Press, 2006, ISBN 1-424-40464-9.

[11] M. Strauss, C. Reynolds, and R. W. Picard, "The handwave bluetooth skin conductance sensor," in *Proceedings of the 1st International Conference on Affective Computing and Intelligent Interaction*, 2005.

[12] E. H. Chi, J. Song, and G. Corbin, ""killer app" of wearable computing: wireless force sensing body protectors for martial arts," in *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM Press, 2004, pp. 277–285.

[13] H. Markus, H. Takafumi, N. Sarah, and T. Sakol, "Chi-ball, an interactive device assisting martial arts education for children," in *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM Press, 2003, pp. 962–963.

[14] P. Haemaelaeinen, T. Ilmonen, J. HoeysniemI, M. Lindholm, and A. Nykaenen, "Martial arts in artificial reality," in *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 2005, pp. 781–790.

[15] V. Henderson, S. Lee, H. Brashear, H. Hamilton, T. Starner, and S. Hamilton, "Development of an american sign language game for deaf children," in *Interaction Design and Children*, June 2005.

[16] Chua, *Training for physical tasks in virtual environments: Tai Chi*, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1191125

[17] T. Starner, B. Leibe, B. Singletary, and J. Pair, "Mind-warping: towards creating a compelling collaborative augmented reality game," in *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*. ACM Press, 2000, pp. 256–259. [Online]. Available: http://portal.acm.org/citation.cfm?id=325864

[18] H. Junker and P. Lukowicz and G. Tröster, "Continuous recognition of arm activities with body-worn inertial sensors," in *Proceedings of the Eighth International Symposium on Wearable Computers*, 2004.

[19] O. Amft, H. Junker, and G. Tröster, "Detection of eating and drinking arm gestures using inertial body-worn sensors," in *ISWC2005: IEEE Proceedings of the Ninth International Symposium on Wearable Computers.*, October 2005, pp. 160–163.

[20] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proceedings of the IEEE International Conference on Data Mining*, 2001.

[21] 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 the Architecture of Computing Systems (ARCS 2006)*, LNCS, Springer-Verlag, 2006.