Gestures are Strings: Efficient Online Gesture Spotting and Classification using String Matching

Thomas Stiefmeier
Wearable Computing Lab
ETH Zürich
Zürich, Switzerland
stiefmeier@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

Context awareness is one mechanism that allows wearable computers to provide information proactively, unobtrusively and with minimal user disturbance. Gestures and activities are an important aspect of the user's context. Detection and classification of gestures may be computationally expensive for low-power, miniaturized wearable platforms, such as those that may be integrated into garments.

In this paper we introduce a novel method for online and real-time spotting and classification of gestures. Continuous user motion, acquired from a body-worn network of inertial sensors, is represented by strings of symbols encoding motion vectors. Fast string matching techniques, inspired from bioinformatics, spot trained gestures and classify them. Robustness to gesture variability is provided by approximate matching efficiently implemented through dynamic programming. Our method is successfully demonstrated by spotting and classifying the occurrences of trained gestures within a continuous recording of a complex bicycle maintenance task. It executes in real-time on a desktop computer with a fraction of CPU time. Only simple integer arithmetic operations are required, which makes this method ideally suited for implementation on body-worn sensor nodes and real-time operation.

Categories and Subject Descriptors

I.5.5 [**Pattern Recognition**]: Implementation; I.5.4 [**Pattern Recognition**]: Application; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

1. INTRODUCTION

Wearable computers, embedded in clothing or seamlessly integrated in devices we carry with us, may revolutionize the way we work, collaborate, learn or share experiences. They aim to support users by providing information proactively, unobtrusively and with minimal disturbance. One key mechanism enabling wearable computers is the recognition of the user's context, such as his location, his social interactions or his gestures and activities [6, 17]. By being "always on" and continuously evaluating the state of the user, they are at the crossroads of personal assistants and pervasive computers, and may provide feedback and information tailored to the user's needs, anytime and anywhere.

Physical activities and gestures are an important aspect of the user's context. By identifying gestures, wearable computers may for instance support workers in industrial environments [27, 28]; they may enhance social interactions [11]; or they may even provide an insight into affective disorders or depression [9]. Gestures can be characterized by the trajectory of the limbs in Cartesian space. These trajectories may be acquired by external absolute trackers (e.g. ultra-sound, vision), but in order to work in unconstrained environments, wearable computers tend to rely on body-worn sensors. Inertial sensors (e.g. accelerometers and gyroscopes) worn on the limbs are commonly used for this purpose. The identification of gestures depends on their nature. Periodic motions are often recognized by classification of signal features [23, 29] with simple means like nearest neighbor classifiers; non-periodic motions may be detected by hidden Markov models (HMMs) [22], which were used to detect intentional gestures from body worn-sensors [4] and from limb motions tracked by vision systems [26, 30].

Real-time and continuous recognition of activities and gestures remains a complex task which is often tackled in a two-step approach: 1) segmentation and 2) recognition. Various segmentation methods have been proposed: Keogh et al. introduced a theoretical framework for the segmentation of time series for data mining [12], Lee et al. proposed a threshold model using HMMs to detect "non-gestures" [14], Wilson et al. discussed the use of the probability of occurrence according to trained HMMs to segment gestures [30] and Junker et al. proposed a segmentation based on closed motions [10]. Ogris et al. investigated motion segmentation based on location information [21], whereas Stäger et al. explored sound-based segmentation [25]. The isolated recognition of the segmented motion is often achieved using HMMs [4, 30]. However, this two-step approach may be computationally expensive.

We propose a single-step approach that does not require explicit motion segmentation to perform online and real-time gesture spotting and classification of continuous user motion. It is related to threshold based HMMs although we argue that the computational requirements are more suited for low-power devices such as described in [24]. Our method relies on very fast string matching techniques applied on strings of symbols representing continuous motion. Strings as an abstraction of motion has also been used in [18]. Despite variability in gestures of identical classes, robust spotting is possible by using approximate string matching that is efficiently implemented through dynamic programming. Our method is also related to dynamic time warping (DTW) [2],[13] but does not require costly arithmetic operations such as divisions, and may be implemented entirely in in-

teger arithmetics. This makes it ideally suited for efficient implementation in micro-controllers such as those available on wearable sensor nodes.

We demonstrate our method by spotting and classifying occurrences of trained manipulative hand gestures within a continuous sequence containing gestures and a predominant amount of "non-gestures" of users performing bicycle maintenance tasks. We evidence its strength both as a method to spot gestures within a continuous stream of data, as well as a classification method. Its capacity to spot gestures may also be used to segment data for traditional isolated classification methods. In Section 2, we describe exact and approximate string matching techniques and how they may be efficiently implemented with dynamic programming. The process of converting limb trajectories in Cartesian space into strings, and how string matching is applied to those, is explained in Section 3. Section 4 describes results of applying our novel method to a real-life bicycle maintenance task. Results are discussed in Section 5 and we conclude this paper in Section 6.

2. STRING MATCHING

String matching techniques are used to support information retrieval and computational biology (e.g. analysis of DNA or protein sequences). It can be applied to any sequence of symbols. We assume that gestures can be represented by strings, as will be shown in Section 3. Following, we introduce the theoretical framework of string matching.

A *string* is a sequence of characters or symbols over a finite alphabet Σ. In general, the string matching problem is to find all occurrences of a given string S, also referred to as the *pattern*, in a larger sequence of symbols, the *text* T. String matching deals with two problems: *exact matching* and *approximate matching*. We consider approximate string matching since it allows to cope with variability of the pattern's occurrences in the text. A good introduction to string matching and an in-depth overview on commonly deployed algorithms is given in [20].

2.1 Approximate String Matching

In order to allow for a certain degree of deviation between string S and potential occurrences in text T, edit operations applied to string S and leading to a substring of T are allowed. The edit operations are *insertion* of a symbol into string S, *deletion* of a symbol from S and the *substitution* of a symbol in S with another symbol. The *edit distance* or *Levenshtein distance* characterizes the difference between two strings [15]. It is the minimum number of edit operations to transform one string into another string. A general form of the edit distance is obtained when dedicated weights are assigned to the available edit operations. Approximate matching consists in finding the substring of T that matches best string S according to the weighted edit distance.

2.2 Algorithm

The weighted edit distance is computed through *dynamic programming*, which allows an efficient implementation. Each computational problem formulated using dynamic programming is based on a recurrence relation. In the case of the weighted edit distance, this relation is established using the computation of $D(i,j)$ which represents the minimum edit operation cost needed to transform the first i symbols of string S_1 into the first j symbols of string S_2. The recurrence relation including insertion and deletion cost d and substitution cost r can be stated the following way (see [8]):

$$D(i,j) = min\,[D(i-1,j)+d, D(i,j-1)+d, \\ D(i-1,j-1)+t(i,j)]$$

Expression $t(i,j)$ is used to handle substitution and equality, where $t(i,j) = 0$ if $S_1(i) = S_2(j)$; otherwise $t(i,j) = r$. The initial conditions of the recurrence relation are: $D(i,0) = i \cdot d$ and $D(0,j) = j \cdot d$. They account for the deletion costs for transforming the first i symbols of string S_1 into the empty string and the insertion costs for transforming the empty string into the first j symbols of string S_2. In order to calculate the weighted edit distance for string S_1 of length n and string S_2 of length m, the recurrence relation needs to be invoked $n \cdot m$ times. As can be seen easily, there are no costly multiplications or divisions; only three addition operations, a comparison and a minimum operation which can be implemented based on comparisons. The computational complexity of the algorithm is $O(nm)$. The memory requirements to store intermediate results in this general version of the algorithm can be stated as $O(nm)$.

2.3 Application of String Matching

The edit distance gives a similarity measure for two strings S_1 of length n and S_2 of length m. In the problem of spotting gestures in a continuous stream of data, string S_1 is the pattern which is to be found in the continuous motion of the user represented by string S_2. The process of converting continuous motion into a string is described in detail in Section 3. Expression $D(n,j)$ is the minimum weighted edit operation cost to align string S_1 into an occurrence (or substring) ending at index j within S_2. The lower the cost, the higher the likelihood of having found a gesture occurrence within the continuous motion of the user.

3. GESTURE SPOTTING AND CLASSIFICATION USING STRING MATCHING

This section describes how string matching is used to spot occurrences of gestures in a continuous stream of data. We consider gestures carried out with hands and refer to them as manipulative gestures or simply *gestures*. For this work, sensor modules placed on the body are used to acquire the trajectory of the right hand. The trajectory is referred to as *continuous motion*. It is aggregated, quantized and encoded using a finite set of symbols resulting in a *motion string*. For each gesture class to be spotted within the motion string, a generic representation is created which is referred to as *template string*. These templates are generated during training as well as statistics are collected (e.g. spotting thresholds). Template strings are continuously matched to the motion string during spotting and classification. The matching cost based on the edit distance is used to spot gestures and classify them. Figure 1 illustrates this procedure. The processing path for template string training is indicated with dashed arrows. Solid arrows depict the path taken by the continuous motion and the motion string in which template string occurrences are spotted and classified.

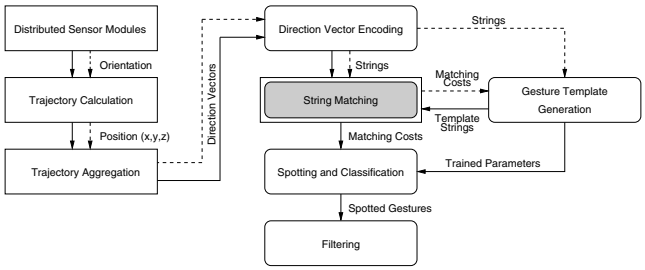

Figure 1: Overview of gesture spotting and classification using string matching

3.1 Trajectory Calculation

A continuous stream of orientation data is acquired from a distributed set of inertial sensor modules mounted on the lower arms, the upper arms and the torso of the body. By combining the modules' orientation data, the relative position of the arms with respect to the torso is computed including the limbs' and the torso's dimensions. Assuming that the motion patterns of the hands are a good indicator for manipulative gestures considered in this work, the relative position of the hands is computed in Cartesian coordinates. A continuous stream of such relative positions is referred to as a *trajectory* in three-dimensional space.

3.2 Trajectory Aggregation

In order to transform trajectories, of which the coordinates have continuous values, to a set of symbols appropriate for string matching, quantization is required. Contiguous positions in the trajectories are first aggregated to form trajectory segments. These segments are then transformed to direction vectors which are encoded by discrete symbols. The direction vectors represent the motion direction within trajectory segments. Two aggregation modes are investigated: (a) temporal aggregation: segments of identical temporal duration $w \cdot T_s$ are aggregated, (b) spatial aggregation: segments of the same spatial distance ϵ are aggregated. T_s stands for the sampling period. These modes define the start and end point of the trajectory segment to be aggregated. End point of segment i is contiguous to start point of segment $i+1$ in terms of positions in a trajectory. The direction vector of a segment is the vectorial difference of the positions of its end and start points.

3.3 Direction Encoding

The direction vectors obtained in the previous steps are quantized. Each direction vector is mapped to an entry of a codebook which contains a finite set of unit direction vectors. The codebook vectors are directionally uniformly distributed. The actual mapping of a given direction vector to one codebook vector is achieved by finding the closest codebook vector in terms of angular distance. Every codebook vector corresponds to a character or symbol. A neutral symbol representing the null vector is also provided.

Figure 2 shows an example trajectory which is segmented using temporal aggregation with $w = 8$. The start and end points of one example segment are marked with circles. The corresponding direction vector is plotted with a bold arrow. In addition, 6 unitary codebook vectors are illustrated corresponding to symbols $\Sigma = \{a,b,c,d,e,f\}$. The 7th symbol g (null vector) is not shown. In this example, the trajectory segment is encoded into symbol e since it represents the closest codebook vector. In Figure 3, the whole process of encoding a motion string, matching template strings with it and calculating the matching costs for two gesture classes is depicted.

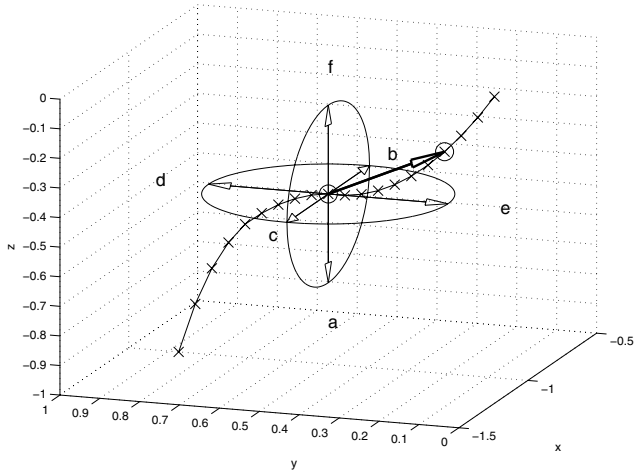

Figure 2: This example shows a trajectory in Cartesian space (samples are marked with X) and one trajectory segment with its corresponding direction vector (bold arrow) and start/end points (circles). A codebook sphere with 6 codebook vectors not including the null vector is overlaid to show the encoding of the segment into symbol *e*.

3.4 Matching Costs

As described in Section 2, the string matching method is based on the computation of weighted edit distances. In this work, the required insertion, deletion and substitution costs are based on the angular relation of two codebook vectors represented by symbols. In case two symbols, i.e. two codebook vectors, are identical, no cost is assigned. Otherwise, a cost based on the angular relationship of the two vectors is associated. We assign an exponentially increasing cost for an increasing angular distance of the two vectors to penalize large angular deviations.

$$f_c(i,j) = \exp\left(\frac{\angle(v_i, v_j)}{100}\right) - 1 \qquad (1)$$

Equation 1 calculates the insertion, deletion and substitution cost for two symbols i and j, which represent direction vectors v_i and v_j, using function $\angle$ to determine the angle in degrees between two vectors.

3.5 Training

Training is carried out in two steps. At first, a template string for each gesture class is created which represents all training instances of that very class. For that, the weighted edit distance is calculated for all combinations of two training instances. The training instance that minimizes the sum of distances to all the other instances is selected as the template string. In a second step, matching cost statistics between the template string and all the training instances of

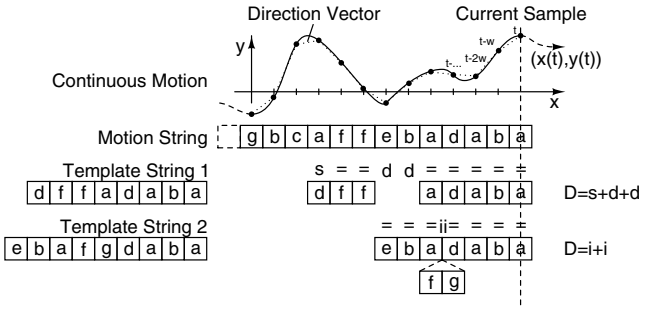

Figure 3: Continuous motion (here in two dimensions) is aggregated into trajectory segments, transformed into direction vectors and encoded into symbols. The resulting sequence of symbols is called *motion string*. Gesture classes are represented by templates which are created during training resulting in *template strings*. These are aligned with the motion string during spotting using approximate string matching which allows matches ('='), insertions ('i'), deletions ('d') and substitutions ('s'). For each gesture class, a threshold acquired during training is applied to the computed matching cost D to spot gesture occurrences.

that class are collected. The mean μ_i and the standard deviation σ_i of the matching cost is computed for each gesture class i. A class-related threshold k_{thr} is then derived the following way:

$$k_{thr,i} = \mu_i + \nu \cdot \sigma_i, \tag{2}$$

where ν is a parameter which needs to be optimized for an individual data set.

3.6 Spotting and Classification

Gesture spotting requires a template string for each gesture class to be spotted. The matching cost for each template string with the motion string (continuous data stream of symbols) is computed resulting in a stream of matching costs for each class. Within these cost streams, local minima are detected. When local minima are below threshold $k_{thr,i}$ for class i (see Equation 2), a spotted occurrence of the particular class i is reported. The local minima being below the thresholds are anticipated to be end points of the spotted occurrences. Therefore, an implicit classification is performed. Due to the particular template string length, the start point of this occurrence can be computed.

3.7 Filtering

Since the spotting and classification is carried out in parallel for all gesture classes, temporal collisions between spotted gestures can occur. These collisions are detected by monitoring the reported gesture occurrences in all classes and checking their overlap. Collisions are resolved by comparing the relative matching costs of colliding spotted gesture occurrences, which is defined as the absolute matching cost of a template string with the continuous stream of symbols divided by the class-dependent threshold $k_{thr,i}$. The occurrence with the lowest relative matching cost is the output of the filtering and thus the result of the classification; all other colliding spotted gestures are discarded.

4. EXPERIMENTAL RESULTS

4.1 Bicycle Maintenance Task

Gesture spotting and classification using string matching in a continuous stream of data has been validated on a data set which consists of manipulative gestures in a realistic setting. This data set comprises 23 different distinguishable maintenance tasks within a bicycle maintenance scenario [21]. Figure 4 shows the bicycle while one of the subjects is performing a task. Orientation data of the torso, the upper and lower arms is acquired using MTx inertial sensor modules from Xsens at a sampling rate of 50 Hz.

Figure 4: Bicycle maintenance scenario

Five tasks of this maintenance scenario have been selected to demonstrate the functionality of the novel method. The data set is partitioned into two parts: For each of the maintenance tasks, a data recording is available which contains between 20 and 25 repetitions of the manipulative gesture. This part is used to train the template strings and the class-dependent thresholds which are used for gesture spotting. The second part of the data set consists of test sequences which contain all 23 gestures in a random order. These sequences are used to validate our method. Each sequence contains 7 occurrences of gestures we wish to detect and 16 other gestures which form the null class in addition to ordinary movements which are not related to any of these 23 gestures. This makes the sequence close to real-life conditions for gesture spotting. To improve the realism even more, the user was asked to clean random bike parts at random time within each test sequence not interrupting the ongoing gestures.

The following manipulative gestures are considered; their number of occurrences per test sequence is given in brackets: *pumping a wheel* (2), *turning pedals* (2), *turning pedals and mark unbalances on back wheel* (1), *open back light and remove bulb* (1) and *insert bulb and close back light* (1). 8 sequences per subject have been recorded which leads to 56 occurrences of gestures to detect. The data of 3 subjects has been processed. In total this adds up to 208 minutes of sequence data for validation.

4.2 Gesture Spotting

Figure 5 shows the matching cost at each time step (i.e. for each encoded symbol) between a training sequence containing 25 repetitions of a gesture and the template string derived from this training sequence. The end points of the training occurrences in the sequence (vertical lines) match

exactly the spotted occurrences, which are all the circles below the dashed horizontal line representing k_{thr}. It can

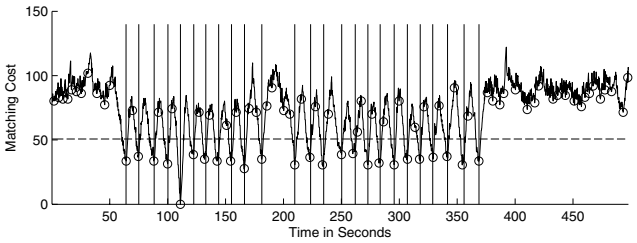

Figure 5: Matching cost on training data (class: *pumping a wheel*, 25 repetitions) using temporal aggregation with $w = 5$ and a codebook size of 7

be clearly seen that data recorded before and after the 25 repetitions of this training sequence are not spotted as gestures (matching cost above threshold). A noise-like characteristic of the matching cost is observed in these ranges. The template string lengths for all gesture classes using spatial aggregation with $\epsilon = 20mm$ (temporal aggregation with $w = 5$) are the following: pumping a wheel 90 (75), turning pedals 89 (56), turning pedals and mark unbalances 69 (43), open back light 44 (73) and close back light 41 (92). For the results presented in this work, we used the following parameters: $\nu = 2.0$, $\beta = 0.5$, $w = 5$ for temporal aggregation, $\epsilon = 20mm$ for spatial aggregation and a codebook size of 7.

Figure 6 illustrates the spotting of two gesture occurrences of the same class in a test sequence comprising 23 different gestures as explained in Section 4.1. In addition to the matching cost, the *pitch angle* of the right upper arm is plotted (upper curve). The matching cost clearly falls below the detection threshold at the end of the two occurrences whose ground truth is indicated by the two shaded areas. Figure 7 shows a close-up on the first occurrence of the considered gesture. It evidences the close match between the spotted location of the end of the gesture and the ground truth which corresponds to the time span in which the arm is performing periodic motions as can be seen in the pitch angle signal.

4.3 Spotting and Classification Accuracy

We use an objective quality measure in order to evaluate the performance of our spotting and classification method. It counts the number of times when a gesture spotted by the string matching method corresponds to the ground truth defined by human labeling of the data set. In order to cope with the temporal inaccuracy of the labeling, on overlap parameter β defines the minimum overlap ratio of the spotted gesture with a ground truth gesture that is required to say that the spotting matches the ground truth. This ratio is based on the length of the ground truth gesture. For $\beta = 0.5$, the overlap needs to exceed 50% of the ground truth length to obtain a correctly spotted gesture. Each spotted gesture is assigned to one of the four following categories: *correct* (the spotted gesture is aligned with the correct corresponding ground truth gesture according to β), *inserted* (the spotted gesture does not correspond to any ground truth gesture), *deleted* (the ground truth gesture has not been spotted) and *substituted* (the ground truth gesture has been spotted but classified incorrectly).

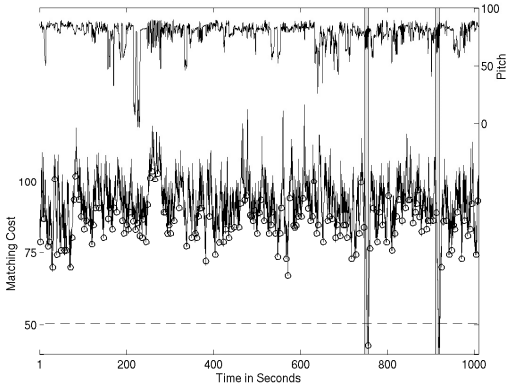

Figure 6: Matching cost of a template string (class: *pumping a wheel*) on a test sequence using temporal aggregation with $w = 5$. Depicted is a whole sequence of 1008 seconds length with two gesture occurrences marked by shaded areas. For both occurrences, the matching cost (lower graph) falls below the corresponding threshold which is shown as a dashed line. In addition, the *pitch angle* of the right upper arm is plotted (upper graph).

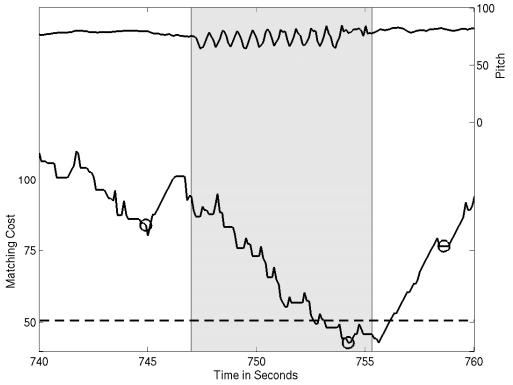

Figure 7: Close-up of the first gesture occurrence (shaded area) as depicted within the whole testing sequence in Figure 6.

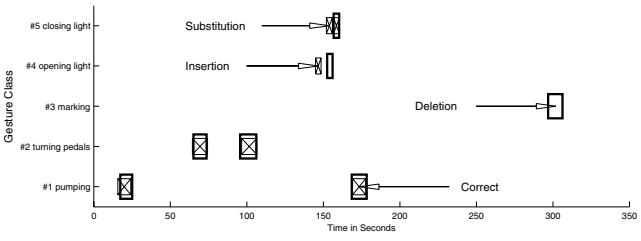

Figure 8: Gestures of 5 classes are spotted and classified in a test sequence of 350 seconds. Spotted gestures are depicted by rectangles with diagonal lines; ground truth gestures by bold rectangles. We find 5 correct, 1 deletion, 1 insertion and 1 substitution.

	Correct Events	Deleted Events	Inserted Events	Substituted Events	Ground Truth Events	Correct Rate	Deletion Rate	Insertion Rate	Substitution Rate
Subject 1	46	4	2	6	56	82.1%	7.1%	3.6%	10.7%
Subject 2	44	11	9	1	56	78.6%	19.6%	16.1%	1.8%
Subject 3	49	7	15	0	56	87.5%	12.5%	26.8%	0.0%
Total	139	22	26	7	168	82.7%	13.1%	15.5%	4.2%

Table 1: Spotting and classification results for different subjects. To achieve these results, spatial aggregation mode with $\epsilon = 20mm$ has been used.

Figure 8 illustrates the spotting and classification results on a whole test sequence. 7 ground truth gestures are included in this sequence of which 5 are spotted and classified correctly. One insertion of gesture class 4 occurs around time 145. A gesture of class 3 is deleted around time 300. A substitution occurs between class 4 and 5 at time 155. In general, the number of correct, deleted and substituted gestures adds up to the number of ground truth gestures. Table 1 shows the spotting and classification results for 24 test sequences from 3 different subjects. On average, 82.7% of all ground truth gestures are spotted and classified correctly. This is an interesting result, since only comparatively low processing effort is required for the string matching algorithms. About 13.1% of the ground truth gestures have been deleted producing errors which cannot be compensated later on. However, the deletion rate is significantly varying among the different subjects. Insertion (15.5%) and substitution (4.2%) errors can be coped with using additional postprocessing, for example an additional classifier.

4.4 Performance

As explained in Section 2, the computational complexity of the approximate string matching algorithm is $O(nm)$ where n represents the length of the template string and m the length of the motion string. The algorithm is applied to the symbol stream (motion string) for each of the t gesture classes to be spotted. Defining the average template string length $\bar{n}$, the required CPU time of an online implementation is proportional to $t \cdot \bar{n}$. The template string lengths and the number of gesture classes are constant during the whole spotting procedure. For performance comparison reasons, three algorithms, which are commonly used for segmentation as the first processing stage of spotting, have been implemented and applied to the same data set: a) SAX [16] approximates a given time series by piecewise constant segments which are encoded into a discrete alphabet , b) SWAB [12] merges contiguous samples of a time series until a cost measure is reached and c) a genetic algorithm (GA) based approach [7] uses evolutionary search to find a proper segmentation. It needs to be emphasized that SAX, SWAB and GA only segment the data stream in our implementation. A second costly processing stage, i.e. a similarity search, is needed for spotting and classification.

Table 2 indicates the CPU time required by each of the algorithms on a Pentium 4 (3GHz, 1GByte RAM) to process various sequences. All algorithms are implemented in Matlab without special optimizations. The execution time of our spotting and classification method is about 12.4% of the duration of the sequences. This means the algorithm is roughly 8 times faster than real-time. The CPU time is distributed among the five key processing blocks (see Figure 1)

Test Sequence	Sequence Duration	CPU Time for String Matching	CPU Time for SAX	CPU Time for SWAB	CPU Time for GA
1	1008	129.7	73.9	1271.8	1750.5
2	532	66.2	21.9	875.5	913.1
3	466	57.7	20.2	824.0	795.6
4	487	60.1	20.6	875.4	856.4
5	434	52.3	14.1	794.9	759.0
6	423	51.8	15.0	812.8	736.1
7	495	60.4	16.7	892.1	848.6
8	378	45.8	10.0	715.1	643.3
Total	4223	524.0	192.4	7061.6	7302.6

Table 2: Test sequence duration and CPU time in seconds for spotting and classification of 8 sequences. For these results, the temporal aggregation with $w = 5$ has been used.

for spatial aggregation with $\epsilon = 20mm$ (temporal aggregation with $w = 5$) as follows: aggregation 1.65% (0.65%), direction vector encoding 1.91% (2.23%), string matching 96.04% (96.41%), spotting and classification 0.36% (0.64%) and filtering 0.04% (0.07%). We expect to reduce the required CPU time considerably by implementing the algorithms in C or C++. SWAB and GA require more than 67% more processing time in our implementation than would be available for a real-time execution on this computer. SAX is about 2.7 times faster than our spotting and classification method, however it does not perform spotting nor classification but only a segmentation of the continuous data stream.

5. DISCUSSION AND FUTURE WORK

We introduced a new method for online gesture recognition which encodes gestures by strings. String matching between a template string and the ongoing motion string is used to spot and classify gestures. This method has the advantage of not requiring any time-consuming preliminary gesture segmentation, in contrast to commonly used methods of isolated gesture recognition, e.g. based on hidden Markov models [10]. In addition, gestures can easily be processed online. Comparing the continuous matching cost of the template strings with trained thresholds indicates gesture occurrences. Spotting (the detection of an occurrence) and classification (knowing which class of gestures occurred) are carried out at the same time.

In this work, we used string matching to classify manipulative gestures. Since the motion string is matched with several template strings in parallel, a gesture may be classified as belonging to several classes when its matching costs are all below the detection thresholds computed during training.

To resolve this, gestures are assigned the class for which the template has the lowest relative matching cost with the occurrence. Although providing the desired filtering, a more sophisticated filter should take into account gesture length as well as the history of previously spotted gestures. Another classifier may be used as a second stage classification method when the string matching method comes with ambiguous classification results, thereby potentially improving the classification accuracy. In particular, string matching may be used as a very fast segmentation method to provide isolated segments for classifiers operating on isolated gestures, such as HMMs. Then string matching is used for segmentation only and it may be optimized accordingly. Instead of trying to maximize correct classification while minimizing insertions, deletions and substitutions, the string matching system may be optimized to mostly minimize the number of deletions. High insertion or substitution rates are not critical, since they would be handled by the second classifier, as long as enough computing power is available to classify (and potentially reject) spotted gestures. Deletions, however, are critical: if a gesture is not spotted by the string matching, this particular occurrence is lost regardless of the second classifier.

Template strings should be a generic representation of the class of gestures they stand for. They are currently generated by selecting the string from the training set of gestures that has the smallest overall matching cost with all the other strings. Thereby we constrain the search of the optimal template string. Since an exhaustive search is computationally prohibitive, another approach may be to synthesize a generic gesture from the training class instances and encode it as a template string. Synthesis may be done by traversing the most likely sequence of states of a hidden Markov model trained on the training gestures of one class. This method is successfully exploited to program robots by imitation [1], [3]. The generation of user independent template strings may be carried out similarly.

Currently, only the trajectory of the right hand is processed. A next step will be to find a way of combining this with the processing of other trajectories which is conceivable on different levels: during trajectory encoding or after performing the spotting in parallel on several trajectories.

A performance comparison of the string matching method to related approaches such as dynamic time warping (DTW) [2],[13] or HMM-based spotting methods [5],[14] remains object of future work. However, the computational complexity of the string matching approach is anticipated to be lower than DTW and HMM. In [13], for each new data sample the whole DTW algorithm has to be performed to find the optimal warping path which is more costly than our string matching approach which only involves few comparisons for each new data sample. In addition, parameters for endpoint detection have to be defined either by hand or in a complex automated way making the DTW approach less generic. As described in [5], the complexity of the HMM-based spotting approach is $O\left(TN^2\right)$ with T equals the number of observation steps and N equals the number of states in the HMM. This is clearly computationally more demanding than the complexity of the string matching approach which accounts for $O\left(nm\right)$, where m corresponds to the number of observed motion symbols and n equals the length of the string template (see Section 2.2).

The current implementation of the spotting and classification was done in Matlab without special optimizations, yet it is about 8 times faster than real-time on a desktop computer. The implementation in particular does not need any complex arithmetic operation (e.g. divisions, trigonometric functions) and can be implemented entirely in integer arithmetic. As a consequence, the string matching method is well suited for an implementation on low-power micro-controller systems. We intend to capitalize on this efficient method by investigating online gesture recognition on low-power sensor nodes.

6. CONCLUSION

In this paper we introduced for the first time a method for online gesture spotting and classification that is based on efficient string matching techniques, similar to those used in bioinformatics. Continuous motion trajectories are encoded as a sequence of symbols representing the direction of motion vectors. Gesture templates (strings describing gestures to be recognized) are matched online with the string of symbols representing the ongoing motion trajectory. An efficient approximate string matching algorithm based on dynamic programming is used to spot gestures in a robust way, allowing variability across gestures of the same class. A high degree of match between a template gesture string and the user motion string indicates a gesture occurrence of the corresponding class.

We demonstrated this new approach by recognizing typical manipulative gestures carried out in a bicycle maintenance setup using the motion trajectories of the worker's right hand. Realistic sequences were recorded with 23 typical bicycle maintenance tasks occurring in a random order and with various null class events in between the worker's maintenance activities. We detected occurrences of five classes of gestures while the remaining worker activities were part of the null class. Results showed that this new method was able to spot trained gestures among continuous recording of motion with a high success rate (correct rate $> 82\%$), while the number of misclassifications remained low. In addition, processing was about 8 times faster than real-time on a desktop computer without any implementation optimizations.

The string matching method is fast and uses only simple integer arithmetic operations (e.g. no costly division operations). Together with its good performance, it is well suited for implementation on low-power sensor nodes that may be placed on the body or integrated into garments for wearable computing. This approach may also serve for fast gesture segmentation and spotting, instead of more costly methods relying on motion dynamics analysis [10] or on hidden Markov models [30, 14].

However, the real strength of string matching for gesture recognition is yet to be realized. We envision that high-level activities composed of sequences of elementary activities and gestures may be spotted and classified in a similar way by introducing string matching operators that apply to groups of symbols in addition to isolated symbols [19], therefore paving the way to hierarchical activity recognition.

7. REFERENCES

[1] A. Billard, Y. Epars, S. Colinon, S. Schaal, and G. Cheng. Discovering optimal imitation strategies. *Robotics and Auton. Systems*, 47(2-3):69–77, 2004.

[2] A. F. Bobick and A. D. Wilson. A state-based approach to the representation and recognition of gesture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1325–1337, Dec. 1997.

[3] S. Calinon and A. Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proc. of the 22nd Int. Conf. on Machine Learning*, 2005.

[4] G. Chambers, S. Venkatesh, G. West, and H. Bui. Hierarchical recognition of intentional human gestures for sports video annotation. In *Proc. of Conference on Pattern Recognition*, pages 1082–1085, 2002.

[5] J. Deng and H. Tsui. An HMM-based approach for gesture segmentation and recognition. In *15th International Conference on Pattern Recognition*, volume 2, pages 679 – 682, September 2000.

[6] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context awareness. Technical Report GITGVU-99-22, Georgia Tech, 1999.

[7] T. C. Fu, F. L. Chung, V. Ng, and R. Luk. Evolutionary segmentation of financial time series into subsequences. In *Proc. of the 2001 Congress on Evolutionary Computation*, pages 426–430, May 2001.

[8] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambrigde University Press, 1997.

[9] J. M. Hausdorff, C.-K. Peng, A. L. Goldberger, and A. L. Stoll. Gait unsteadiness and fall risk in two affective disorders: a preliminary study. *BMC Psychiatry*, 4(39), 2004.

[10] H. Junker, P. Lukowitz, and G. Tröster. Continuous recognition of arm activities with body-worn inertial sensors. In *Proc. of the Int. Symposium on Wearable Computers*, pages 188–189, 2004.

[11] M. Kanis, N. Winters, S. Agamanolis, A. Gavin, and C. Cullinan. Toward wearable social networking with iBand. In *CHI '05 - Human factors in computing systems*, pages 1521 – 1524, 2005.

[12] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proc. of the Int. Conf. on Data Mining*, pages 289–96, 2001.

[13] M. H. Ko, G. West, S. Venkatesh, and M. Kumar. Online context recognition in multisensor systems using dynamic time warping. In *Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 283–288, Dec. 2005.

[14] H.-K. Lee and J. H. Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):961–973, Oct. 1999.

[15] V. I. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. 1:8–17, 1965.

[16] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discorery*, pages 2–11, 2003.

[17] P. Lukowicz, H. Junker, M. Staeger, T. von Bueren, and G. Troester. WearNET: A distributed multi-sensor system for context aware wearables. In G. Borriello and L. Holmquist, editors, *Proc. of the 4th Int. Conf. on Ubiquitous Computing*, pages 361–370, Heidelberg, Sept. 2002. Springer.

[18] D. Minnen, T. Starner, I. Essa, and C. Isbell. Discovering characteristic actions from on-body sensor data. In *Proc. of IEEE International Symposium on Wearable Computing*, pages 11–18, Oct. 2006.

[19] S. Muthu Muthukrishnan and S. Cenk Sahinalp. Simple and practical sequence nearest neighbors with block operations. In *Proc. of the 13th Annual Symposium on Combinatorial Pattern Matching*, pages 262–278, 2002.

[20] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings*. Cambrigde University Press, 2002.

[21] G. Ogris, T. Stiefmeier, H. Junker, P. Lukowicz, and G. Tröster. Using ultrasonic hand tracking to augment motion analysis based recognition of manipulative gestures. In *Proc. of IEEE International Symposium on Wearable Computing*, pages 152–159, Oct. 2005.

[22] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, Feb. 1989.

[23] C. Randell and H. Muller. Context awareness by analysing accelerometer data. In *Proc. 4th International Symposium on Wearable Computers*, pages 175–176, 2000.

[24] D. Roggen, N. B. Bharatula, M. Stäger, P. Lukowicz, and G. Tröster. From sensors to miniature networked sensorbuttons. In *Proc. of the 3rd Int. Conf. on Networked Sensing Systems*, pages 119–122, San Diego, CA, 2006. Transducer Research Foundation.

[25] M. Stäger, P. Lukowicz, and G. Tröster. Implementation and evaluation of a low-power sound-based user activity recognition system. In *Proc. of the 8th International Symposium on Wearable Computers*, pages 138–141, Los Alamitos, CA, 2004. IEEE Computer Society Press.

[26] T. Starner, J. Weaver, and A. Pentland. Real-time American sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375, 1998.

[27] T. Stiefmeier, C. Lombriser, D. Roggen, and G. Tröster. Event-based activity tracking in work environments. In *Third International Forum on Applied Wearable Computing*, March 2006.

[28] T. Stiefmeier, G. Ogris, H. Junker, P. Lukowicz, and G. Tröster. Combining motion sensors and ultrasonic hands tracking for continuous activity recognition in a maintenance scenario. In *10th IEEE International Symposium on Wearable Computers*, October 2006.

[29] P. Veltink, H. Bussmann, W. de Vries, W. Martens, and R. Van Lummel. Detection of static and dynamic activities using uniaxial accelerometers. *IEEE Transactions on Rehabilitation Engineering*, 4(4):375–385, 1996.

[30] A. D. Wilson and A. F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):884–900, Sep. 1999.