Limited-Memory Warping LCSS for real-time low-power pattern recognition in wireless nodes

Daniel Roggen[1], Luis Ponce Cuspinera[1], Guilherme Pombo[1], Falah Ali[1], and Long-Van Nguyen-Dinh[2]

[1] Sensor Technology Research Centre, University of Sussex, United Kingdom
[2] Wearable Computing Laboratory, ETH Zurich, Switzerland
daniel.roggen@ieee.org

Abstract. We present and evaluate a microcontroller-optimized limited-memory implementation of a Warping Longest Common Subsequence algorithm (WarpingLCSS). It permits to spot patterns within noisy sensor data in real-time in resource constrained sensor nodes. It allows variability in the sensed system dynamics through warping; it uses only integer operations; it can be applied to various sensor modalities; and it is suitable for embedded training to recognize new patterns. We illustrate the method on 3 applications from wearable sensing and activity recognition using 3 sensor modalities: spotting the QRS complex in ECG, recognizing gestures in everyday life, and analyzing beach volleyball. We implemented the system on a low-power 8-bit AVR wireless node and a 32-bit ARM Cortex M4 microcontroller. Up to 67 or 140 10-second gestures can be recognized simultaneously in real-time from a 10Hz motion sensor on the AVR and M4 using 8mW and 10mW respectively. A single gesture spotter uses as few as 135μW on the AVR. The method allows low data rate distributed in-network recognition and we show a 100 fold data rate reduction in a complex activity recognition scenario. The versatility and low complexity of the method makes it well suited as a generic pattern recognition method and could be implemented as part of sensor front-ends.

Keywords: Activity Recognition; Wearable Sensing; Streaming pattern spotting; Distributed Recognition; Machine Learning; Event Processing

1 Introduction

Spotting patterns in noisy signal streams is important in many sensor network applications [26], such as monitoring integrity of structures [12]; predicting crop needs [23]; or recognizing human activities from wearable or ambient sensors nodes [1], which is our motivation. Activity recognition is used in adaptive smart homes [18] and in wearable smart assistants [17]. In general, multiple networked nodes must be fused to increase accuracy [28] or resilience [20]. In order to minimize energy use and wireless bandwidth, processing should be distributed on the nodes so that only events are sent at low data rate for data fusion [25,9,14,2]. This requires efficient local pattern recognition on the nodes in the first place.

In order to recognize complex patterns (hereafter *motifs*) in noisy sensor signals we present and evaluate a microcontroller-optimized *Limited-Memory* and *Warping* Longest Common Subsequence (LM-WLCSS) implementation of the WarpingLCSS algorithm analyzed offline in [16]. The resulting system allows a real-time streaming execution in memory constrained nodes. It has low computational complexity and uses only integer operations. It allows to dilate or contract the motif to accommodate for variations in the sensed system dynamics, such as human variability. LM-WLCSS has a high specificity to the target motif which allows to spot subtle activities. The sensitivity-specificity trade-off can be adjusted with a single parameter. Low-complexity training is possible on the node, which enables e.g. personalization of activity models at run-time. The method has a defined low latency, which allows use in critical applications. LM-WLCSS can process raw sensor signals or signal features which makes it applicable to scenarios beyond wearable sensing. The method can be used for distributed pattern recognition in sensor nodes by performing local recognition on individual nodes and combining these decisions in a central node, thus leading to significant reduction in network bandwidth.

2 Related work

Spotting patterns in noisy signals has been extensively studied for activity and gesture recognition with wearable devices [3] and the principles generalize to other domains. A common approach combines *segmentation* (e.g. with a sliding window), *feature computation* on that segment, and *classification* of the features into pre-defined classes [1]. Features can be computationally complex and enough memory must be available to store the sensor data corresponding to the longest pattern to spot. This can be a constraint in sensor nodes[3]. With high sample rate, careful optimization is required to meet memory-performance tradeoffs [21], or powerful microcontrollers must be used, e.g. with hardware FPU for EMG analysis [4]. Code optimizations reduce CPU usage but are worthwhile only for general purpose algorithms, as this takes a lot of effort. For instance, hidden Markov models can be implemented in fixed-point arithmetic [27]. Template matching methods compare the sensor signal with a motif resulting in a matching score. Dynamic Time Warping (DTW) allows to dilate or contract the motif to accomodate for signal variability and was used in activity recognition [6,11]. Algorithms based on longest common subsequence were suggested in a sliding-window and a warping form (WarpingLCSS) for online activity recognition and outperformed DTW with noisy data [16,15]. WarpingLCSS computational cost is bound to linear order of the template size and and memory is bound to quadratic order of the template size. However, in previous works it was implemented in floating point and evaluated offline. DTW and WarpingLCSS approaches are both computationally light thanks to dynamic programming implementations and have a simple training process.

[3] The commonly used TMote Sky has 10KB RAM. With a 3D accelerometer and gyroscope sampled at 100Hz and 16 bit, the maximum activity length is 8 seconds.

In a sensor network bandwidth should be minimized. Complex higher-level patterns across multiple nodes can be inferred from lower level events broadcasted by the nodes using fuzzy logic [14], decision fusion [28], meta-classifier [2], sparsity classifier [25]. This can be supported by software frameworks [9]. Another approach is to rely on signal processing techniques such as compressed sensing to reduce bandwidth by exploiting signal statistics [8]. Sparse representations decompose the sensor signal along an optimized basis and also allow to reduce bandwidth as well as improve classification performance. The power usage of a recent implementation was 2W on a dual-core ARM A9 [24].

3 Limited-Memory Warping LCSS Recognition System

We introduce a microcontroller-friendly system to spot motifs in real-time within noisy streaming sensor signals. The system is based on a *Limited Memory* and *Warping* Longest Common Subsequence algorithm (LM-WLCSS), introduced and evaluated offine in [16] as WarpingLCSS[4].

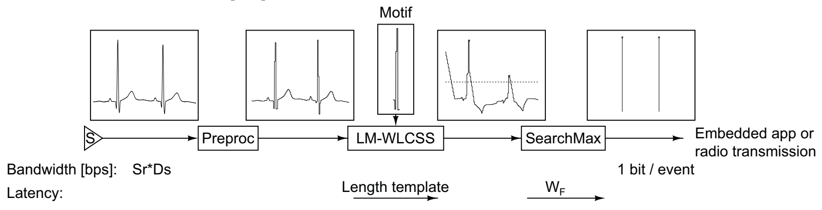


Fig. 1. The sensor data is acquired at Sr Hz and optionally pre-processed with downsampling, feature computation and quantization. LM-WLCSS computes the instantaneous matching score with the motif. Online local maximum search find scores above a detection threshold. This yields an event (1 bit) each time a motif is detected.

The overall pattern recognition system is illustrated in figure 1. The sensor is sampled with sample rate Sr and word length Ds and optionally pre-processed (e.g. by downsampling, computing signal features, or quantization). Afterwards, LM-WLCSS computes the instantaneous matching score between the pre-processed sensor data and the motif: the higher the score, the closer the pre-processed signal is to the motif. Finally, a local maxima search looks for matching scores above an acceptance threshold Thd, which indicates that the motif of interest has been spotted in the sensor signal. At this stage, a single bit or timestamp indicates that a pattern has been spotted. This can be used locally on the node or sent over radio for fusion with detectors on other nodes.

We refer to $\mathcal{S}(i)$ as the ith sample from the sensor (i.e. the input data stream), $\mathcal{T}(j)$ as the jth sample from the motif, and N_T the length of the the motif. The next two subsections describe LM-WLCSS and the local maximum search.

3.1 Limited-Memory Warping Longest Common Subsequence

LM-WLCSS can be efficiently implemented with dynamic programming by solving the problem of matching a shorter motif and a shorter stream and keeping

[4] The update equation is modified from the original work to address edge issues.

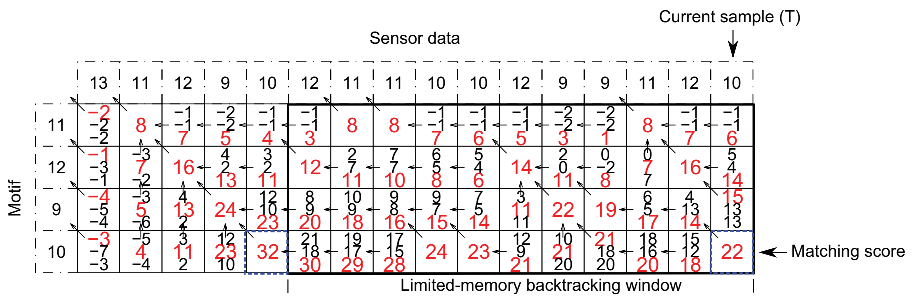


Fig. 2. LM-WLCSS computes the matching score between a motif of length 4 and data coming from a sensor. $R = 8$, $p = 1$, $\epsilon = 0$. The red bold value in the cells is $\mathcal{M}(j, i)$: a single value indicates a match between motif and sensor data; 3 values indicate a mismatch and the 3 possible scores before the *max* operation in equation 1. The last line $\mathcal{M}(N_T, i)$ is the matching score between the motif and the sensor data at time i. A local search of score maxima shows two maxima with score 32 (a perfect match) and 22 at the current sample. The backtracking variable $\mathcal{B}$ is represented by the arrow between cells. Backtracking from the perfect match shows that the motif is aligned with the sensor data without warping. Warping is illustrated when backtracking from the current sample: the motif is dilated and aligned against 9 sensor samples. As a new sample is acquired, a column would be added on the right to compute the updated matching score and backtracking. The limited-memory implementation stores only the last column to update the matching score, and the backtracking is limited in time. Thus, LM-WLCSS is a constant memory algorithm.

intermediate results in memory (see [22] for the classical, non-warping, LCSS). We define $\mathcal{M}(j, i)$ the matching score between the first i samples of the stream and the first j samples of the motif. Thus, $\mathcal{M}(j, i)$ can be computed as follows:

$$\mathcal{M}(j,i) = \begin{cases} 0 & \text{if } i \leq 0 \text{ or } j \leq 0 \\ \mathcal{M}(j-1,i-1) + R & \text{if } |\mathcal{S}(i) - \mathcal{T}(j)| \leq \epsilon \\ max \begin{cases} \mathcal{M}(j-1,i-1) - P \cdot (\mathcal{S}(i) - \mathcal{T}(j)) \\ \mathcal{M}(j-1,i) - P \cdot (\mathcal{S}(i) - \mathcal{T}(j)) \\ \mathcal{M}(j,i-1) - P \cdot (\mathcal{S}(i) - \mathcal{T}(j)) \end{cases} & \text{if } |\mathcal{S}(i) - \mathcal{T}(j)| > \epsilon \end{cases} \tag{1}$$

R is a reward added to the matching score when two samples match. In case of mismatch a penalty proportional to the mismatch between samples scaled by P is applied. A tolerance ϵ allows approximate matches. Warping occurs in case of mismatch with the *max* operation selecting one of three options: accepting a mismatch between one sample from the data stream and the motif (line 1); repeating one element of the data stream (i.e. contracting the motif on line 2); or repeating one element of the motif (i.e. dilating the motif on line 3).

$\mathcal{M}(N_T, i)$ indicates the matching score between the entire motif and the sensor data at time i. We consider that the motif has been found in the sensor data when a local maxima in $\mathcal{M}(N_T, i)$ is found above a trained acceptance threshold. This indicates the end-time of the match. As the algorithm allows for motif warping, the start-time of the match is found by *backtracking* from the end-time, using a backtrack variable $\mathcal{B}(j,i)$ that indicates which option was selected in the assignment of $\mathcal{M}(j,i)$ in equation 1. Figure 2 illustrates how LM-LCSS matches a motif against the sensor data in a matrix representation of $\mathcal{M}$ and $\mathcal{B}$, and how to find the start and end times of the match.

```
**Input**: sample: the current sensor data
**Output**: score: the resulting matching score
/* Limited-memory backtracking window                                    */
$B(1...N_T, 1...W_B - 1) \leftarrow B(1...N_T, 2...W_B)$;
/* Initialization                                                        */
$mu \leftarrow 0$; /* Score in the upper cell                            */
$mul \leftarrow 0$; /* Score in the upper-left cell                      */
for $j \leftarrow 1$ to $N_T$ do /* Update the matching score            */
    $ml \leftarrow M(j)$; /* Score in the left cell                      */
    if $|sample - T(j)| < \epsilon$ then /* sample matches the motif     */
        $score \leftarrow mul + R$;
        $B(j, W_B) \leftarrow 0$;
    else /* mismatch                                                     */
        $t = p \cdot |sample - T(j)|$;
        $score, midx = max(mul - t, mu - t, ml - t)$; /* Returns the maximum of the
        arguments and its 0-based index                                  */
        $B(j, W_B) \leftarrow midx$
    end
    $mul \leftarrow ml$;
    $mu \leftarrow score$;
    $M(j) \leftarrow score$;
end
```

Fig. 3. This function updates the matching score whenever a new sample is acquired. M is a vector of size N_T, B is the backtracking window of size $N_T \times W_B$, and T is a the motif of size N_T; these state variables are kept in-between calls to this function.

Implementation memory can be minimized by realizing that it is not necessary to store the entirety of $\mathcal{M}(j,i)$; instead, only the last column of $\mathcal{M}(j,i)$ is required to compute $\mathcal{M}(j,i+1)$ when the next sample is acquired. Finding the start point of the match requires the backtracking variable $\mathcal{B}(j,i)$. However, application knowledge can be used to provide an upper bound on the amount of warping allowed. Therefore, instead of storing the entirety of $\mathcal{B}(j,i)$, a backtracking window of size W_B can be defined to keep only the most recent (closest to current time T) entries of $\mathcal{B}(j,i)$. The resulting algorithm (figure 3) is called each time a new sensor sample is received to update the matching score.

3.2 SearchMax

Each time the score is updated the function represented in figure 4 is called to find whether the score is a local maxima above above a threshold. This algorithm keeps data storage to a minimum and deals with the issue that signals carrying noise produce many local extrema. The algorithm looks for a local maxima in a sliding window without the need to store that window. The algorithm compares current score (S) with the last score (P) in order to determine whether there is a positive slope. When this is true a flag is set and the maximum value is stored (Max); a counter (K) is used to determine whether the stored value is the maximum within a window (W_F). The the maxima is above a detection threshold Thd the function returns indicating that a motif may have been spotted.

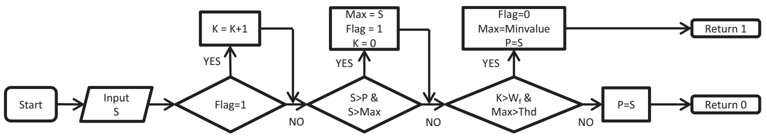


Fig. 4. Algorithm returning whether the current matching score is a local maxima above a threshold within a sliding window of size W_F.

3.3 Embedded training

Training consists of defining the motif and the threshold Thd. Embedded training is possible, for instance for activity recognition. In training mode, the node indicates when it is ready for the user to demonstrate a gesture, e.g. by emitting a sound. The user demonstrates the gesture and the node continuously records the motion sensor data until the user stops moving. The recorded data is the gesture motif. This process can be repeated to evaluate the variability between the gestures and define an optimal detection threshold. One motif (e.g. the first recording) is selected, and the matching score between that motif and the subsequent recordings is computed. In order to spot all the gestures in that dataset, Thd should be equal to the lowest obtained matching score. However to be robust to outliers setting $Thd = \mu_{score} - n \cdot \sigma_{score}$ allows to adjust the sensitivity-specificity tradeoff of the algorithm in a with n[5]. Training with cross-validation can be done offline for better multiparametric optimization [16,15].

3.4 Embedded implementation

We implemented the system in C. A timer interrupt is used to sample the sensor data at regular intervals. The entire pattern spotting process can be executed in the timer interrupt as the processing time is predictable. Alternatively, the timer interrupt can store the data in a buffer, which is processed from the main program code later.

The indexing variables looping through the motif and backtracking window are 16-bit on the AVR and 32-bit on the M4. The entries in the backtracking

[5] This expression allows to approximate a suitable threshold in an online implementation; with n=2-4 our experiments showed good sensitivity-specificity tradeoffs. A better training uses cross-validation but may require too much memory to hold all patterns to be suitable for the sensor node.

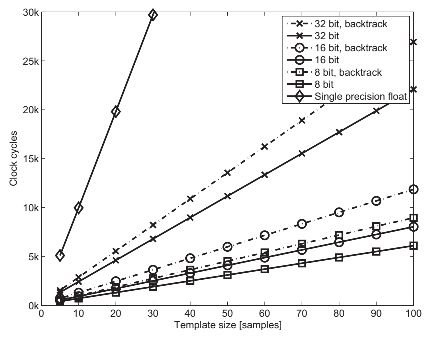
window are 8-bit. For benchmarking, we used different word size for the samples, matching scores, penalty and reward: 8-bit, 16-bit and 32-bit integers and single precision (32-bit) float. While smaller bit-width may be preferred, there is a lower limit defined by the matching score range. The maximum matching score is equal to $N_T \cdot R$. The minimum matching score is a negative value that depends on the incoming sensor data, N_T and P. As the data distribution can only be statistically characterized, there must be enough room to hold a "large" negative value, otherwise the scores may wrap around. In our implementation we scaled the sensor readings and parameters to ensure no wrap around ever occurred. Alternatively saturation arithmetic could be used, but it is much slower.

When using integer arithmetic the ratio of R to P can be selected to approximate a floating point implementation (such as the offline version presented in [16] which fixes $R = 1$ and assumes $0 < P < 1$). We implemented the backtracking array as a circular buffer, thus avoiding memory moves in algorithm 3. Thus the algorithm speed is independent of the size of the backtracking window. The backtracking window is an optional feature: when deactivated, the start point of the match cannot be found but the memory used is significantly reduced. We show in section 5 that backtracking may not be needed for many spotting applications. We stored the templates in RAM. However if the motif is static or trained infrequently, it could be stored in Flash to free up more RAM.

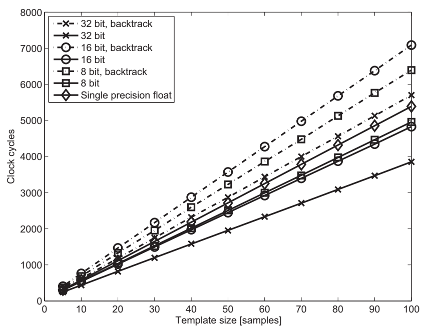
4 Technical evaluation

We characterize the system on two platforms. The first is a custom 8-bit Atmel AVR motion sensor node [19]. It is 44mm×25mm×17mm node with Bluetooth (BlueNiceCom III), a 3D ADXL330 accelerometer, a 2D IDG650 gyroscope and an ATmega1284P microcontroller at 8MHz (see fig 7 left). The AVR has hardware supports for 8-bit multiplications, 128KB of program Flash, and 16KB of RAM. GCC 4.8.1 with O2 optimization is used to compile the system. The second platform is a STM32F4DISCOVERY board with a 32-bit STM32F407 ARM Cortex M4 microcontroller with 1MB of program Flash, 192KB of RAM and a hardware single-precision floating point unit. The microcontroller uses the external crystal with the PLL set to generate an 8MHz CPU frequency. GCC 4.8.4 with O2 optimization is used to compile the system using the thumb2 instruction set. Benchmarking was done using internal timers. The timer resolution was 128μS on the AVR node and 1ms on the M4 board. All benchmarks ran at least one second to minimize measurement error. A serial link (over UART and USB for the AVR node, SWO on the M4 board) is used to report the timings.

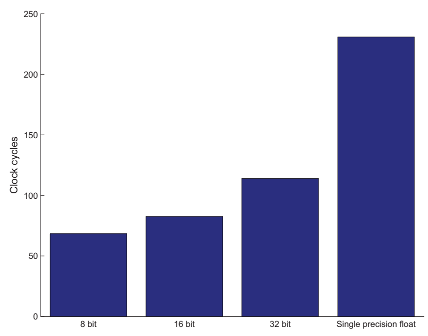
We benchmark individually LM-WLCSS, SearchMax and their combination. The reward and penalty parameters or the range of the motif and sensor data have no influence on speed. Benchmark results are presented in figure 5. Note that the algorithm is linear in $O(N_T)$ in time *for each sample*. Smaller templates allow faster execution, however for very small templates the function call overhead appears in the benchmark. The AVR is faster with smaller word sizes; however 16-bit is an ideal operating point as shown in section 5. The ARM is faster with 32-bit word size, as smaller arithmetic operations must be emulated.



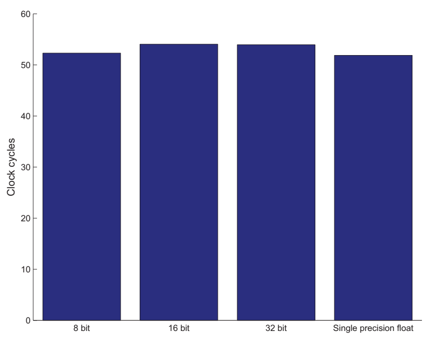
(a) AVR: clock cycles LM-WLCSS



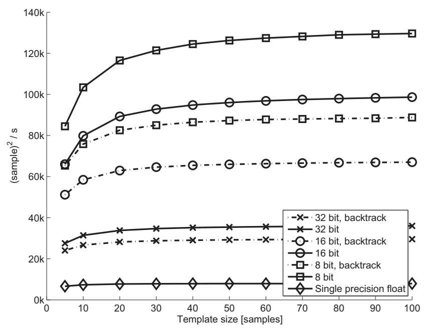
(b) M4: clock cycles for LM-WLCSS



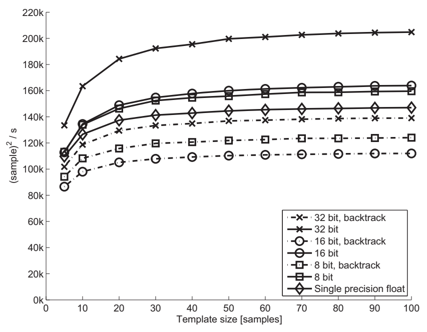
(c) AVR: clock cycles for SearchMax



(d) M4: clock cycles for SearchMax



(e) AVR: S^2/s at 8MHz



(f) M4: S^2/s at 8MHz

Fig. 5. (a,b) show the number of clock cycles for the execution of the LM-WLCSS algorithm only; (c,d) show the number of clock cycles for SearchMax only; (e,f) show the overall system performance when LM-WLCSS and SearchMax are combined in samples^2/second at 8MHz. This unit is the product of the motif length by the maximum sample rate. It is asymptotically a constant. The AVR does 130K sample^2/second at 8MHz. This means it can sustain a sample rate of 1300Hz with a motif of length 100, or 130Hz with a motif of length 1000. The benchmark was performed with different motif sizes; with small motifs the performance decreases due to function call overheads.

The RAM usage can be derived from the algorithm description. The *state variables* can be statically allocated. LM-WLCSS requires memory to store the limited-memory backtracking window $\mathcal{B}$ and the latest column of $\mathcal{M}$. The RAM used of for state data is thus: $N_T \cdot ws + N_T \cdot W_B$ with ws the word size in bytes. If backtracking is disabled, the RAM used for state data is only: $N_T \cdot ws$ SearchMax requires only 5 state variables, regardless of the size of the SearchMax window W_F. A few additonal *working variables* are needed (e.g. mu, mul, ml in algorithm 3), but this is constant and small in contrast to the memory used for the state variables. The compiler may even optimize them out with registers. Consider a 32-bit implementation with a motif of length 30 (i.e. allowing to spot a pattern of 1 second with 30Hz sensor sample rate, which is typical in activity recognition) allowing to find the starting point of the pattern in the data stream even if the pattern is twice slower than the original. Then $W_B = 60$, $N_T = 30$, and the memory needed is: $30 \cdot 4 + 60 \cdot 30 = 1920$ bytes. Note that in section 5 we demonstrate successful spotting without relying on backtracking. If backtracking is disabled, the memory used is only $N_T \cdot ws = 120$ bytes.

In table 1 we report the program size for LM-WLCSS and SearchMax in bytes computed based on the disasembly of the executable. In the float implementation, an additional library for floating point operations is required. Its size is estimated by adding all the functions dealing with floats in the executable. The M4 float implementation is significantly more compact due to the hardware FPU. As few as 434 (16-bit on AVR) and 284 (32-bit on M4) bytes of code are required for the full system with backtracking.

	Platform	8 bit	8 bit, bt	16 bit	16 bit, bt	32 bit	32 bit, bt	float
LM-WLCSS	AVR	186	246	234	310	468	534	578 (+860)
	M4	176	200	184	222	140	180	192 (+3622)
SearchMax	AVR	92		124		194		222 (+860)
	M4	106		110		104		118 (+3622)

Table 1. Program memory (Flash) usage in bytes for LM-WLCSS (top) and SearchMax (bottom). The floating point implementation requires in addition a floating point library, whose estimated size is indicated in parenthesis.

The latency of the system is defined by the length of the motif and the size of the SearchMax window W_F. The maximum matching score is reached once the *end* of the motif is identified in the data stream (see fig. 2). Thus, shorter templates reduce the latency of the system, but may decrease its specificity. The SearchMax window avoids detecting multiple events when in reality only one occured (e.g. with noisy data). The ideal W_F is selected experimentally, but it can be much smaller than the motif size (e.g. 5-10 samples in section 5).

5 Pattern spotting Examples

We illustrate the versatility of the algorithm on 3 examples of pattern recognition typical of wearable sensing and activity recognition. The system parameters have been selected to illustrate the algorithm behavior, not necessarily to achieve the optimal performance. We purposely show a variey of sample rate and motif lengths. In all the examples, we use a 16-bit implementation of the system.

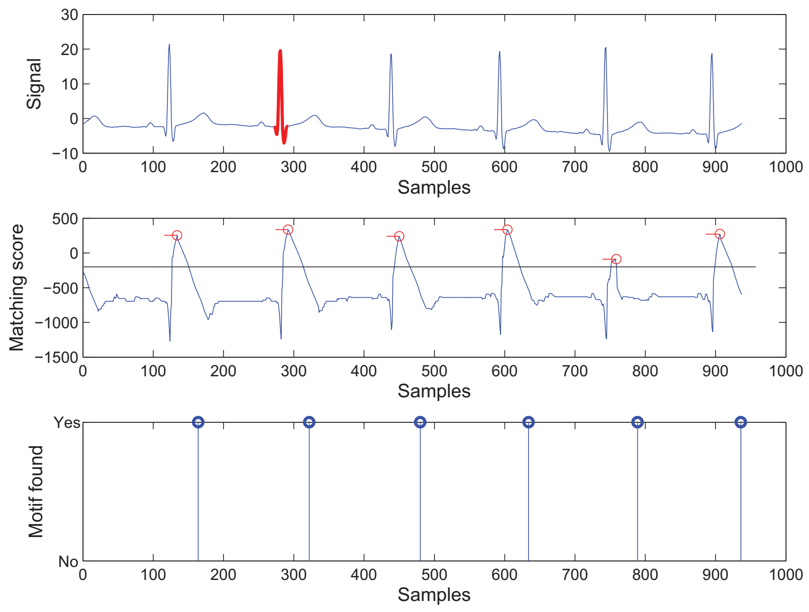
Physiological signal analysis We first illustrate spotting physiological patterns. The top plot in figure 6(a) illustrates the ECG (v2) of a healthy subject sampled at 200Hz. A motif of 20 samples (100ms) is defined around the QRS complex. The system parameters are: $R = 16$, $P = 8$, $\epsilon = 2$, $W_F = 30$, $W_B = 100$, $Thd = -200$. The middle plot shows the matching score which increases above the detection threshold (horizontal line) when the QRS complex is observed and decays as unrelated data is observed. The second last heart beat appears slightly different and only just passes above the threshold. This shows how the threshold can control the sensitivity-specificity tradeoff of the algorithm. A lower threshold would guarantee to spot all the heart beats, but a higher threshold may be desired to spot anomalies in the QRS complex. The lower plot shows the overall latency of the system and indicates the effective time at which the QRS complex is detected. The motif is found some time after the peak in the matching score (controlled by W_F), and the peak occurs when the *end* of the motif is matched against the signal.

The Pan-Tompkins algorithm [10] is the de-facto method to spot the QRS complex. It is based on filtering, derivation, squaring, integration and thresholding, with numerous optimized embedded implementations of the initial algorithm. In comparison LM-WLCSS is very competitive: it offers very low complexity (N_T multiplications per sample), and provides more flexibility than Pan-Tompkins-based methods, as the motif can be adjusted. For instance, it could be used to biometrically identify the user of a device by the ECG shape.

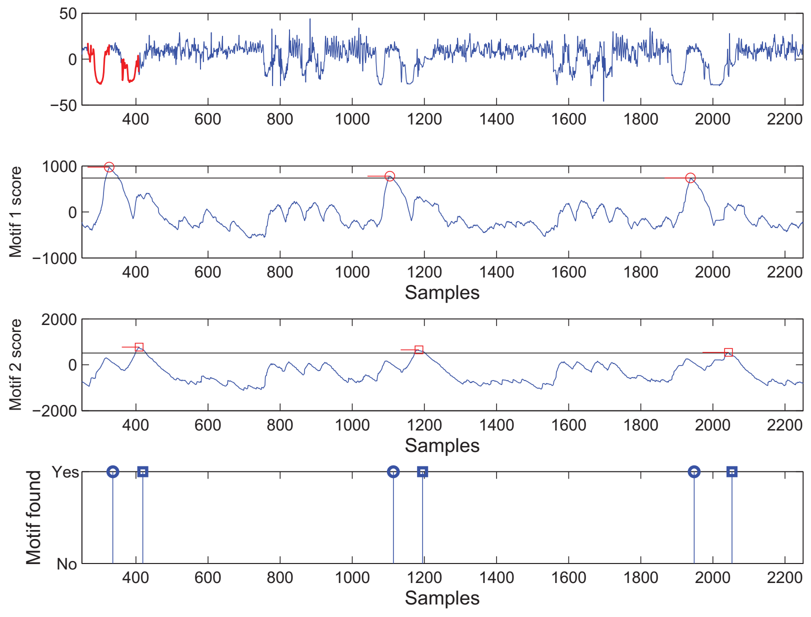
The AVR achieves 65KS^2/s at 8MHz for a motif of 20 samples (fig. 5). A dedicated system could run the CPU at only 490KHz to spot the QRS complex. This would allow operation at 1.8V, using 360uW of power (extrapolated from the datasheet) for the signal processing.

Recognition of everyday activities We show the recognition of everyday activities from an arm-worn accelerometer based on the “Drill run” of the OPPORTUNITY dataset, which is a recognized highly challenging benchmarking dataset as reported in [7]. A person performs 20 repetitions of a scripted but realistic sequence of everyday activities in a home environment, including opening/closing doors/windows/drawers, cleaning a table, drinking, etc. We evaluated LM-WLCSS on the detection of very similar gestures: drinking from a cup while seated, and drinking from a cup while standing or walking. Only one axis of an acceleration sensor node on the dominant lower arm is used. It is quantized in the range -64 to 63 and downsampled from 30 Hz to 10Hz. The WM-LCSS parameters are: $R = 16$, $P = 1$, $\epsilon = 5$, $W_F = 10$, $W_B = 100$. The drink sitting and standing motifs are 61 and 48 samples respectively. The detection threshold were optimized by cross-validation.

Figure 6(b) shows a closeup of 3 of the 20 repetitions of the activity sequence. The sensor data appears very noisy, and there are only very subtle differences between the two motifs. Drinking seated can be recognized more robustly than drinking standing, as the peak in matching score in the first case is more marked,



(a) Spotting of QRS complex in ECG



(b) Recognition of everyday activities

Fig. 6. (a) Detection of the QRS complex in ECG recordings with LM-WLCSS. Top: original signal and highlighted motif. Middle: matching score, threshold, and identified local maxima. Bottom: effective detection of the QRS complex, with the additional latency of SearchMax. **(b)** Detection of "drinking seated" (2nd plot) and "drinking standing" (3rd plot) gestures from a 1 axis acceleration channel on the lower arm.

which allows to set a higher detection threshold. Nevertheless, the algorithm is able to spot and distinguish the two kinds of subtly different gestures.

Assuming a motif length of 100 and 10Hz sample rate (i.e. gestures of up to 10 seconds), the AVR can do $67\text{KS}^2/\text{s}$ with the 16-bit backtracking implementation. This allows to recognizing 67 different gestures in real-time at 10Hz using 8mW (3.3mA at 2.4V with the internal 8MHz RC oscillator). Alternatively, one gesture could be recognized with the CPU running at 120KHz only. At 1.8V and with the internal 128KHz RC oscillator this gives 135uW for a single gesture spotter. On the M4, the fastest backtracking implementation does $140\text{KS}^2/\text{s}$. This allows to recognize 140 gestures at 10Hz with the CPU at 8MHz with power consumption of 10mW (3.3V, 3mA at 8MHz according to the datasheet). Power decreases by more than 50% by disabling backtracking in all cases.

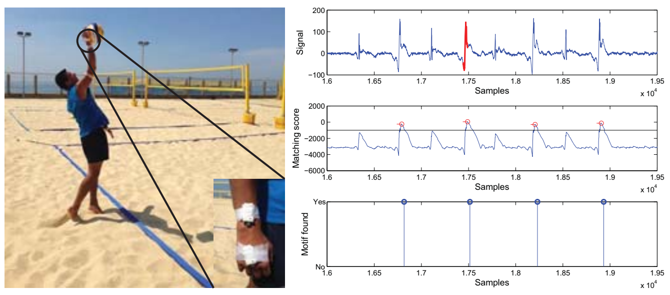


Fig. 7. Sensors placement for beach volleyball serve analysis (left) and detection of selected Beach Volleyball serves from a forearm gyroscope sensor (right).

Beach Volleyball We show the recognition of beach volleball serves from one gyroscope placed on the forearm as shown in Figure 7. The player was asked to serve several times from different parts of the court and varying power, and data was collected from 64 serves using the AVR-based sensor node described in section 4. We observed that the player's routine before serving included a smack on the ball to remove the sand on it, therefore the LM-WLCSS algorithm was used to analyze the data and evaluate the discrimination of both events. Selected data is shown in Figure 7, showing the serve template (of size 50 samples) and the smack before serving. The LM-WLCSS parameters are $R = 1$, $P = 1$, $\epsilon = 10$, $Thd = -1000$, $W_F = 25$. Using a single axis of the gyroscope, we recognized the servers with only 1 false positive and 20 false negatives. These results are promising considering the variability of serves [5] and that there has not been any particular optimization for this application.

6 Extensions and discussion

When using LM-WLCSS on raw acceleration readings the system is sensitive to sensor displacement and rotation. Other template matching methods suffer from the same issue. One solution is to apply the method on features derived from the acceleration, such as the acceleration magnitude which is rotation independent.

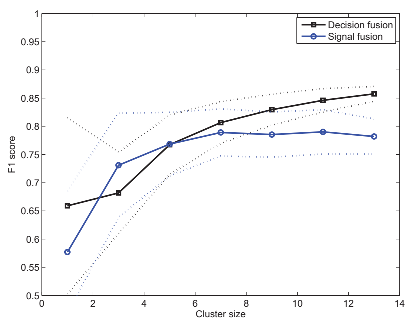


Fig. 8. Average performance (standard deviation in dashed lines) from an exhaustive evaluation of clusters of size 1 to 13 nodes on the upper limbs in an everyday activity recognition task (17 activity classes). In signal fusion, the nodes stream raw data to a central node that runs LM-LWCSS. In decision fusion, nodes run locally LM-LWCSS; when recognizing a pattern they send an event to a central node that fuses the individual decisions with majority voting. Decision fusion requires much lower datarate.

Displacement can be handled by first detecting on-body sensor placement in order to select adequate motifs for recognition [13].

The main behavioral difference between this implementation and the WarpingLCSS work in [16] results from the use of integer arithmetic. We found out that an 8-bit implementation is generally inadequate for acceleration data, however 16-bit (or more) are adequate for the scenarios presented here.

Memory is reduced by decreasing the size of the backtracking window. This limits the maximum dilation (but not the contraction) of the motif if the start point of the match is desired. If the start point of the match is not desired the backtracking window can be eliminated altogether.

We showed the recognition of at most 2 simultaneous patterns in the drink example. This can be extended to more motifs[6]. One challenge is that several motifs may be simultaneously spotted if the data is very noisy. This can be addressed by a conflict resolution as presented in [16], or by ranking the activity likelihood using the matching score which can be used with decision fusion.

Results were obtained from a single sensor channel. Multiple sensor channels (e.g. 3D acceleration) can be combined before being processed by LM-WLCSS with K-means clustering [16,15], or by modifying the sample matching to a vector Euclidian distance (e.g. to handle 3D acceleration). This *signal fusion* can be used to fuse multiple channels on a single node, or across multiple nodes when they stream their raw signals to a central node doing K-means clustering and running LM-WLCSS. Alternatively, nodes can perform individual local pattern recognition with LM-LWCSS and send events to central node which performs *decision fusion*, for example using majority voting [28] or meta-classifiers [2]. This leads to very low data rates, as radio transmission only occurs when a

[6] Memory usage with multiple motifs is the sum of the memory needed to recognize each motif individually.

pattern is recognized by a node and needs as few as $log_2(C)$ bits, with C the number of classes.

In figure 8 we compare signal fusion and decision fusion performance on the recognition of 17 distinct activities (340 gestures in total) from the OPPORTUNITY "Drill run" for clusters of nodes of various size [7]. We consider 13 nodes on the upper limbs. Each node is a 1 axis acceleration or rate of turn sensor. We assess all combinations of clusters of size N out of the 13 nodes and report averages and standard deviation. In signal fusion, all nodes in the cluster stream raw data (16-bit per sample) to a central node at 10 Hz which performs a k-means clustering (k=20) before applying LM-LCSS with 17 motifs. The total bandwidth is $10 \cdot 16 \cdot N$ bps. In decision fusion, each node of the cluster performs local classification and send 5 bit each time an event is recognized. We consider a worst case setup for decision fusion, which assumes no null-class. The total bandwidth is $5 \cdot N$ bit per gesture; given the average duration of a gesture is 3.8 seconds, the bandwidth for decision fusion is $1.3 \cdot N$ bps. This leads to a reduction of bandwidth by 2 orders of magnitude (from $160 \cdot N$ bps to $1.3 \cdot N$ bps), while keeping similar recognition performance in this scenario. As expected, performance increases with the size of the cluster as more information is available to recognize the user's activities. Although it appears that decision fusion outperforms signal fusion we cannot make such a general statement from the limited amount of data used.

The simplicity of the LM-WLCSS codepath makes it is suitable for silicon-level implementation, for instance based on a multiply-and-accumulate unit to to execute in n clock cycles the algorithm 3 for a motif of length n. A silicon implementation would allow ultra-low power pattern spotting, and could be included in sensor frontends of microcontrollers.

LM-WLCSS allows a training by demonstration that is important as ever more assisted living and smart assistant applications require *personalization* to handle human variability. It also allows a simple control of the sensitivity and specificity tradeoff with Thd, which can be adapted depending on the application need (e.g. to spot any drink even v.s. only specific drink events). An increase in Thd increases the specificity of the method and decreases its sensitivity. The system parameters can be optimized by cross-validation [16,15].

7 Conclusion

We have shown a motif matching method to spot patterns in noisy signal streams suitable for real-time execution with low-latency on sensor nodes. We presented two variants of the algorithms: one that simply spots the moment that a motif is observed in the sensor data, the other is capable of *backtracking* to find the start time of the match, which indicates how much the motif has been "warped". The first implementation uses only as much RAM as the length of the motif and is sufficient to spot patterns in a wide range of applications, as demonstrated in this paper with 3 scenarios involving 3 different kinds of sensors.

With backtracking, we reach a performance of $67\text{KS}^2/\text{s}$ for a 16-bit implementation on an 8-bit AVR microcontroller, and $140\text{KS}^2/\text{s}$ on a 32-bit Cortex M4 microcontroller. For a motif of length 100 (e.g. a gesture of maximum 10 seconds at 10Hz) the AVR and M4 at 8MHz can recognize respectively 67 and 140 motifs in real-time from a 10Hz sensor, consuming respectively 8mW and 10mW and using as few as 434 or 284 bytes of code for the full system. The AVR can realize a single gesture spotter using only 135uW. In a distributed activity recognition scenario, LM-WLCSS allows a bandwidth reduction by 2 orders of magnitude with identical performance to a signal fusion approach. This is especially interesting to support context awareness in opportunistic sensing scenarios.

LM-WLCSS is a generic algorithm which we demonstrated to be useful in a wide range of pattern recognition scenarios. This makes LM-WLCSS well suited for distributed in-network pattern recognition, which could be implemented in next generation smart accessories (smart-watches, smart-bracelets), smart environments, and more generally in the Internet of Things. Future work may include silicon implementation to further reduce power in smart nodes.

References

1. Avci, A., Bosch, S., Marin-Perianu, M., Marin-Perianu, R., Havinga, P.: Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In: Proc Int Conf on Architecture of Computing Systems. pp. 1–10 (2010)
2. Bahrepour, M., Meratnia, N., Havinga, P.: Sensor fusion-based event detection in wireless sensor networks. In: Mobile and Ubiquitous Systems. pp. 1–8 (2009)
3. Bao, L., Intille, S.S.: Activity recognition from user-annotated acceleration data. In: Pervasive Computing: Proc. of the 2nd Int'l Conference. pp. 1–17 (2004)
4. Benatti, S., Farella, E., Benini, L.: EMG embedded HMI for smart garments. In: Atelier of Smart Garments and Accessories Workshop at Ubicomp (2014)
5. Buscà, B., Moras, G., Peña, J., Rodríguez-Jiménez, S.: The influence of serve characteristics on performance in men's and women's high-standard beach volleyball. Journal of Sports Sciences 30(3), 269–276 (2012)
6. C., P., Ploetz, T., Olivier, P.: A dynamic time warping approach to real-time activity recognition for food preparation. In: Proc 1st Int Joint Conf on Ambient Intelligence. Springer (2010)
7. Chavarriaga, R., Sagha, H., Calatroni, A., Digumarti, S., Millán, J., Roggen, D., Tröster, G.: The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. Pattern Recognition Letters 34, 2033–2042 (2013)
8. Chen, Z., Ranieri, J., Zhang, R., Vetterli, M.: DASS: Distributed adaptive sparse sensing. arXiv:1401.1191 (1013)
9. Fortino, G., Guerrieri, A., Bellifemine, F.L., Giannantonio, R.: SPINE2: Developing BSN applications on heterogeneous sensor nodes. In: Proc. IEEE Symposium on Industrial Embedded Systems (2009)
10. J., P., Tompkins, W.J.: A real-time QRS detection algorithm. IEEE Transactions on Biomedical Engineering 32(3) (1985)
11. Kale, N., Lee, J., Lotfian, R., Jafari, R.: Impact of sensor misplacement on dynamic time warping based human activity recognition using wearable computers. In: Proc Wireless Health (2012)
12. Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., Turon, M.: Health monitoring of civil infrastructures using wireless sensor networks. In: 6th Int Symp on Information Processing in Sensor Networks. pp. 254–263 (2007)

13. Kunze, K., Lukowicz, P.: Dealing with sensor displacement in motion-based onbody activity recognition systems. Proc. 10th Int. Conf. on Ubiquitous computing (2008)
14. Marin-Perianu, M., Lombriser, C., Amft, O., Havinga, P., Tröster, G.: Distributed activity recognition with fuzzy-enabled wireless sensor networks. In: Int Conf on Distributed Computing in Sensor Systems (2008)
15. Nguyen-Dinh, L.V., Calatroni, A., Tröster, G.: Robust online gesture recognition with crowdsourced annotations. Journal of Machine Learning Research 15, 3187–3220 (2014)
16. Nguyen-Dinh, L.V., Roggen, D., Calatroni, A., Tröster, G.: Improving online gesture recognition with template matching methods in accelerometer data. In: Proc 12th Int Conf on Intelligent Systems Design and Applications. pp. 831–836 (2012)
17. Patel, S., Park, H., Bonato, P., Chan, L., Rodgers, M.: A review of wearable sensors and systems with application in rehabilitation. Journal of NeuroEngineering and Rehabilitation 9(21) (2012)
18. Rashidi, P., Cook, D.J.: The resident in the loop: Adapting the smart home to the user. IEEE Transactions on Systems, Man, and Cybernetics journal, Part A 39(5), 949–959 (2009)
19. Roggen, D., Bächlin, M., Schumm, J., Holleczek, T., Lombriser, C., Tröster, G., Widmer, L., Majoe, D., Gutknecht, J.: An educational and research kit for activity and context recognition from on-body sensors. In: Proc. IEEE Int. Conf. on Body Sensor Networks (BSN). pp. 277–282 (2010)
20. Sagha, H., Bayati, H., Millán, J.d.R., Chavarriaga, R.: On-line anomaly detection and resilience in classifier ensembles. Pattern Recognition Letters 34(15), 1916–1927 (2013)
21. Stäger, M., Lukowicz, P., Perera, N., von Büren, T., Tröster, G., Starner, T.: SoundButton: Design of a Low Power Wearable Audio Classification System. In: Proc of the 7th International Symposium on Wearable Computers. pp. 12–17. IEEE Computer Society Press, Los Alamitos, CA (2003)
22. Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.: Indexing multi-dimensional time-series with support for multiple distance measures. In: Proc 9th ACM SIGKDD Int Conf on Knowledge discovery and data mining. pp. 216–225. ACM, New York (2003)
23. Wark, T., Corke, P., Sikka, P., Klingbeil, L., Guo, Y., Crossman, C., Valencia, P., Swain, D., Bishop-Hurley, G.: Transforming agriculture through pervasive wireless sensor networks. IEEE Pervasive Computing Magazine 6(2), 50–57 (2007)
24. Wei, B., Yang, M., Shen, Y., Rana, R., Chou, C.T., Hu, W.: Real-time classification via sparse representation in acoustic sensor networks. In: Proc 11th ACM Conf on Embedded Networked Sensor Systems. No. 21 (2013)
25. Yang, A.Y., Jafari, R., Sastry, S.S., Bajcsy, R.: Distributed recognition of human actions using wearable motion sensor networks. Journal of Ambient Intelligence and Smart Environments 1, 1–13 (2009)
26. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Computer Networks 52(12), 2292–2330 (2008)
27. Zappi, P., Farella, E., Benini, L.: Hidden markov models implementation for tangible interfaces. In: Intelligent Technologies for Interactive Entertainment. pp. 258–263 (2009)
28. Zappi, P., Roggen, D., Farella, E., Tröster, G., Benini, L.: Network-level power-performance trade-off in wearable activity recognition: a dynamic sensor selection approach. ACM Transactions on Embedded Computing Systems 11(3) (2012)