A Modified HMM Forward Algorithm for an Embedded Motion Type Classification

W. Kurdthongmee
Division of Computer Engineering, School of Engineering and Reources, Walailak University, Nakorn-si-thammarat, Thailand
Email: kwattana@wu.ac.th

***Abstract*—Hidden Markov model is well-known for its application in temporal pattern recognition. Its disadvantages are its computational expensive and very prone to numerically underflow. The focus of this paper is on the forward algorithm which computes the probability of a particular output sequence with respect to the HMM parameters. To make it feasible to implement on an embedded system, we propose a modified forward algorithm that makes use of integer only representation and operations. The outcome of these modifications is integrated into a motion type classification system used for elderly monitoring; 7 motion types with 2×2 transition and 8 emission probabilities, on a low cost embedded system based on a 32-bit ARM Cortex-M0+. The system is capable to perform with comparable classification correctness to the ordinary and the scaling coefficients algorithms. It outperforms the ordinary ones by taking 10 percent of time, 91 percent of code size and 54 percent of memory. It is capable of forcing the processor to sleep the longest with only 3.1ms execution time per second (8.7 and 3.1 percent of the ordinary and the scaling coefficients algorithms). This makes it more suitable for implementation on an embedded system.**

***Index Terms*—hidden markov model, embedded system, motion type classification, forward algorithm, elderly monitoring system**

I. Introduction

A hidden Markov model [1], or HMM for short, is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. A HMM can be considered as the simplest dynamic Bayesian network. The mathematics behind the HMM was developed by L. E. Baum and coworkers. It is closely related to an earlier work on optimal nonlinear filtering problem (stochastic processes) by Ruslan L. Stratonovich, who was the first to describe the forward-backward procedure. In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a HMM, the state is not directly visible, but the output which is dependent on the state is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective "hidden" refers to the state sequence through which the model passes, not to the parameters of the model. Even if the model parameters are known exactly, the model is still "hidden".

A HMM is especially known for its application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

A HMM can be considered as a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

By definition [2], [3], a HMM has a set of hidden states, Q, an output alphabet (observations), O, transition probabilities, A, output (emission) probabilities, B, and initial state probabilities, Π. The current state is not observable. Instead, each state produces an output with a certain probability (B). Usually the states, Q, and outputs, O, are understood, so an HMM is said to be a triple: (A, B, Π). Eq. (1) represents the formal definition of the HMM terms.

$$\begin{aligned} Q &= \{q_i\}, i = 1,..., N \\ A &= \{a_{ij} = P(q_{j,t+1} \mid q_{i,t})\} \\ O &= \{o_k\}, k = 1,..., M \\ B &= \{b_{ik} = b_i(o_k) = P(o_k \mid q_i)\} \\ \Pi &= \{\pi_i = P(q_{i,t=1})\} \end{aligned} \tag{1}$$

There are 3 canonical problems to solve with a HMM:

1) Given the model parameters (A, B, Π), compute the probability of a particular output sequence. This problem is solved by the forward, which is the focus of this paper, and Backward algorithms.
2) Given the model parameters (A, B, Π), find the most likely sequence of (hidden) states which could have generated a given output sequence. This problem is solved by the Viterbi algorithm and Posterior decoding.
3) Given an output sequence, find the most likely set of state transition and output probabilities. This problem is solved by the Baum-Welch algorithm.

Manuscript received April 15, 2014; revised June 30, 2014.

doi: 10.12720/ijsps.2.2.83-90

In practice, the Baum-Welch algorithm is used to find a set of unknown parameters, (A, B, Π), of a HMM given different input sequences during the model training stage. The algorithm can be applied on a computer system with unlimited processing power and resources. Once all the set of parameters have already been created, the forward algorithm is then used to find the probability of a particular observation sequence O with respect to the set of parameters. The model whose parameters give rise to the maximum probability is best matched to the observation sequence O. Many applications require implementing the forward algorithm on a platform with limited resources and less powerful; i.e. an embedded system platform. The drawbacks of the forward algorithm for the embedded system platform are its computational expensive and very prone to numerically underflow. It is the aim of this paper to present a modified forward algorithm that makes use of integer representation and operations.

Automatic motion type classification is very useful in many application areas. The characteristics of the motion types and patterns can be used as an indicator of one's mobility level, latent chronic diseases and aging process [4]. The motion types can be employed to further make a decision if one is at risk; i.e. the motion type or the transition between the motion types may be risky and is likely to cause a fall or it is a fall motion pattern that requires an immediate attention. Alternatively, the motion types may be useful as an indication to request for a close observation/attention; i.e. a jogging is higher risk and requires a close attention than a normal walk especially for elderly people. The indication may be used to support the feature of a video surveillance system for monitoring elderly people. With respect to these examples of application areas in combination with the requirement to automate monitoring of elderly people as a result of "aging society", the demands for an automatic motion classification have been increased. To observe and make relation to motion types, either an acceleration sensor or video system has been widely accepted as useful.

The objective of this research is to design and develop a real-time automatic motion type classification and detection of fall motion pattern given the motion parameters sampled from the body attachment motion sensors (gyroscope and accelerometer). The HMM is employed to create a set of probabilistic models which are expected to be specific to each motion pattern. This means that given a motion pattern as an input for a set of probabilistic models, only a model whose internal representations is the most similar one will response with the highest probability. This is equivalent to saying that the HMM is used to recognize and classify motion patterns and differentiate between normal motion patterns and a fall motion pattern. As the HMM is computational expensive, in order to attain the real-time performance of the classification and detection algorithm, the alternative implementation of the HMM will be pursued. The algorithm is aimed to implement on an embedded system platform with available internal resources.

The organization of this paper is as follow. In Sec. II, the literature survey on the topic of the forward algorithms is given. An integer representation and operations forward algorithm is then detailed in Sec. III. To illustrate the usefulness of the proposed algorithm, it is implemented as a motion type classification system on a low cost embedded system platform. This is described in Sec. IV. Finally, the paper is concluded in Sec. V.

II. Previous Work

In this section, the background of the ordinary forward algorithm is given. Then, the previously proposed approaches to rectify the drawbacks of the algorithm are followed. Let $\alpha_t(i)$ be the probability of the partial observation sequence $O_t = \{o_1, o_2, ..., o_T\}$ to be produced by all possible state sequences that end at the i-th state.

$$\alpha_t(i) = P(o_1, o_2, ..., o_T \mid q(t) = q_i) \tag{2}$$

Then, the unconditional probability of the partial observation sequence is the sum of $\alpha_t(i)$ over all N states.

The forward algorithm is a recursive algorithm for calculating $\alpha_t(i)$ for the observation sequence of increasing length t. First of all, the probabilities for the single-symbol sequence o_1 are calculated by use of Eq. (3).

$$\alpha_1(i) = p_i b_i(o_1),\ i = 1, ..., N \tag{3}$$

Then, the recursive formula given in Eq. (4) is applied.

$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(j) a_{ji}\right] b_i(o_{t+1}) \qquad i = 1, ..., N \quad t = 1, ..., T-1 \tag{4}$$

$\alpha_T(i)$ is eventually resulted after iterating Eq. (4) for $T-1$ times. Finally, the required probability is obtained by summing $\alpha_T(i)$ over all states (Eq. (5)).

$$P(o_1, o_2, ..., o_T) = \sum_{j=1}^{N} \alpha_T(j) \tag{5}$$

When implementing a HMM, floating-point underflow is a significant problem. It is apparent that when applying the forward algorithm to long sequences the extremely small probability values causes underflow on most machines. As the forward algorithm sums probability values, it is not a viable solution to apply log-operation to the values in order to avoid underflow. The most common solution to this problem is to use "scaling coefficients"' that keep the probability values in the dynamic range of the machine, and that are dependent only on t.

The scaling coefficients forward algorithm rectifies the numerical underflow by scaling down all $\alpha_t(i)$ in Eq. (4) in every iteration [4], [5]. The scaling factor should only depend on the current time index t, but be independent of

the state i. A commonly used scaling scheme for computing forward variables is by introducing the scaling factor c_t where:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{6}$$

and replace $\alpha_t(i)$ with the numerical underflow safe $\hat{\alpha}_t(i)$ which is defined by Eq. (7).

$$\hat{\alpha}_t(i) = c_t \alpha_t(i) \tag{7}$$

By doing these, the equation to calculate the probability is changed to be in the form:

$$P(o_1, o_2, ..., o_T) = \exp\left(\sum_{i=1}^{N} \log(\hat{\alpha}_T(i)) \right) \tag{8}$$

Although, the scaling coefficients forward algorithm gives rise to the exact value of probability of the partial observation sequence, it is necessary to calculate the scaling factor c_t and use it to divide the $\alpha_t(i)$ of all states i, Eq. (8), in every iteration t. In addition, it requires invoking computational intensive functions of logarithm and exponential in the last step. This makes it unsuitable for embedded system implementation.

According to our literature review, it has found that researchers in the field have continuously attempted to make it possible to operate the forward algorithm on a real-time basis. This is accomplished by porting the ordinary or scaling coefficients algorithms either to the GPU (Graphic Processing Unit) [4], [6] or the FPGA (Field Programmable Gate Array) [5], [7]. Different architectures and implementations have shown that the acceleration is impressive and significant as they make use of the parallelism within the target platforms. While the objective of our research is also to speed-up the overall operation of the forward algorithm, our target platform is focused to be an embedded system with a low cost and performance.

III. An Integer Representation and Operations Forward Algorithm

A. The Proposed Forward Algorithm

Our proposed algorithm is aimed to be implementable on an embedded system platform. It is designed to rely on using integer representation and operations. The overall side effect of performing these causes the final results to only approximate the exact value; i.e. Eq. (5). By selecting the suitable set parameters, to be mentioned shortly, the deviation is kept minimal. With respect to the proposed algorithm, during the preprocessing stage, the HMM parameters \$A, B\$ and Π are scaled up to their nearest integer values by multiplying them with the scaling factors A_f, B_f and Π_f, respectively. All scaling factors are chosen to be the power of two in order to simplify the probability, Eq. (5), calculation at the termination stage of the algorithm. As the right shifting operation can replace the complex division.

Input;
$\{a\} \leftarrow \{A\} << a_f;$
$\{b\} \leftarrow \{B\} << b_f;$
$\{\pi\} \leftarrow \{\Pi\} << \pi_f;$
Output;
P_n and P_d: nominator and denominator of the probability;

```
/* Initialization                                  */
for i ← 1 to N do
    α_1(i) ← π_i × b_i(o_1);
end
/* Induction                                       */
n_d ← 0;
for t ← 1 to T − 1 do
    for j ← 1 to N do
        z = 0;
        for i ← 1 to N do
            z ← z + (α_t(i) × a_ij);
        end
        α_{t+1}(j) ← z × b_j(o_{t+1});
    end
    /* Scaling down if required to do so.
       */
    if Max({α_{t+1}}) > BN then
        for j ← 1 to N do
            α_{t+1}(j) ← α_{t+1}(j) >> d_f;
        end
        /* Update the number of scaling
           down time.                               */
        n_d ← n_d + 1;
    end
end
/* Termination                                     */
P_n ← 0;
for j ← 1 to N do
    P_n ← P_n + α_T(j);
end
P_d ← (a_f(T − 1) + b_f T + π_f) − d_f n_d;
```

Algorithm 1. An integer representation and operations based hmm forward algorithm

Our proposed algorithm is presented in Algorithm (1). During the operation of the algorithm, Eq. (3) and Eq. (4) are performed as usual. By scaling up the HMM parameters, the floating-point underflow is not possible. All parameters can be represented by integer number and the complex floating point operations can be substituted by integer operations. It could, however, cause the intermediate results to overflow. The additional steps taken after applying Eq. (4) are to check for the possible overflow of any $\alpha_t(i)$. If such case occurs, all $\{\alpha_{t+1}(i)\}$ are required to scale down by a factor of D_f and the number of time that the D_f is applied to $\{\alpha_{t+1}(i)\}$ is updated. Let n_d be the number of time that the D_f is applied. It is initialized to be 0 at the preprocessing stage. The D_f and n_d will be used later to calculate the probability $P(o_1, o_2, ..., o_T)$ at the termination stage of the algorithm.

At the termination stage of the algorithm, the probability $P(o_1, o_2, ..., o_T)$ is equal to Eq. (5) multiplied by $D_f^{n_d}$ and divided by $A_f^{n_d-1} B_f^{n_d} \Pi_f$. This can be described by Eq. (9):

$$P(o_1, o_2, ..., o_T) = \sum_{j=1}^{N} \alpha_T(j) \frac{D_f^{n_d}}{A_f^{T-1} B_f^T \Pi_f} \tag{9}$$

As mentioned earlier that A_f, B_f and Π_f are chosen to be the power of two. That is to say, their values are 2^{a_f}, 2^{b_f} and 2^{π_f}, respectively.

$$\begin{aligned} a_f &= \log_2(A_f) \\ b_f &= \log_2(B_f) \\ \pi_f &= \log_2(\Pi_f) \end{aligned} \tag{10}$$

Similarly, D_f is also chosen to be the power of two which is 2^{d_f} where $d_f = \log_2(D_f)$. If the exact probability is required, the division operation can be substituted by the right shifting operation whose number of time is given in Eq. (11):

$$\frac{2^{d_f n_d}}{2^{a_f(T-1)} 2^{b_f T} 2^{\pi_f}} = \frac{2^{d_f n_d}}{2^{a_f(T-1)+b_f T+\pi_f}} \tag{11}$$

It is noticed that the denominator of Eq. (11) is always more than the nominator. This is equivalent to saying that the times to perform the right shifting operation is equal to:

$$P_d = (a_f(T-1) + b_f T + \pi_f) - d_f n_d \tag{12}$$

With the introduction of the P_d, the first term on the right hand side of Eq. (9) can now be called the P_n whose form is given in Eq. (13):

$$P_n = \sum_{j=1}^{N} \alpha_j(T) \tag{13}$$

In practice, we have found that it is not necessary to find the exact value of probability; i.e. P_n / P_d. Only the maximum probability from different HMMs is considered. The determination of the maximum probability that is free from division is proposed in Sec. III-C.

B. Validation of the Proposed Algorithm

From Eq. (9), it can be seen that the probability of our proposed algorithm is the product between the ordinary one and the ratio between the scaling up and down factors. During the operations, the ordinary intermediate probability value is totally scaled down by $D_f^{n_d}$. That is to say it is scaled down n_d times by D_f. The intermediate probability value is also scaled up $T-1$ and T times by A_f and B_f, respectively. In addition, it is scaled up by π_f once during the initialization stage. The scaling down factors and the number of time they operate result in the denominator term in the equation. With the selections of A_f, B_f, Π_f and D_f to be the power of two, the multiplication with the scaling up factors and division with scaling down factor, if required to perform, can simply be replaced by the simple to perform right and left shifting operations, respectively.

C. The Proposed Algorithm for Determination of the Maximum Probability

The determination of the maximum probability given the set of $P(o_1, o_2, ..., o_T)$ is very important. This is performed in order to find the best matching model; i.e. the model whose parameters gives rise to the maximum probability with respect to the observation sequence $O_t = \{o_1, o_2, ..., o_T\}$. For instance, in our application the HMMs are trained with the captured acceleration data on a motion type by motion type basis. The model whose probability is the maximum corresponds to the queried motion type with respect to the output $O_t = \{o_1, o_2, ..., o_T\}$. In this section, we present the details of the algorithm for determination of the maximum probability from a set of probabilities represented in the form given in Eq. (14):

$$\begin{aligned} P_n &= \{P_{n,1}, P_{n,2}, ..., P_{n,k}\} \\ P_d &= \{P_{d,1}, P_{d,2}, ..., P_{d,k}\} \end{aligned} \tag{14}$$

Input;
$\{P_n\}$ and $\{P_d\}$;
Output;
I_{max};
/* If all members of the denominator set are equal, return the member the nominator set whose value is the maximum. */
if $Equal(\{P_d\})$ then
  $I_{max} \leftarrow Arg(Max(\{P_n\}))$;
else
  /* Size of the set ($\{P_n\}$ or $\{P_d\}$ */
  $k \leftarrow Size(\{P_d\})$;
  /* Find the minimum denominator of probabilities from the denominator set. */
  $P_{d,min} \leftarrow Min(\{P_d\})$;
  /* Create a temporary array of k members. All members are initialized to be 0. */
  $T \leftarrow \{0, .., 0\}$;
  for $j \leftarrow 1$ to k do
    if $P_{d,j} == P_{d,min}$ then
      $T_j \leftarrow P_{d,j}$;
    end
  end
  $I_{max} \leftarrow Arg(Max(\{T\}))$;
end

Algorithm 2. The algorithm for determination of the index of the maximum probability

The algorithm for determination of the maximum probability is shown in Algorithm 2. First of all, the algorithm checks if all members of the P_d are equal. If yes,

the index i where $i \in \{1,2,....,k\}$ whose P_i is the maximum is the index of the maximum probability. The value of the maximum probability is, therefore, $\frac{P_{n,i}}{2^{P_{d,i}}}$ or $P_{n,i} >> P_{d,i}$. Otherwise, the algorithm searches for the minimum $P_{d,\min}$ within P_d. Then, the candidate set for determination of the maximum probability T is created and all its members are initialized to be zero. The members of T are replaced by P_j where $j \in \{1,2,....,k\}$ whose $P_{d,j}$ are equal to $P_{d,\min}$. Finally, it can simply be concluded that the index j where $j \in \{1,2,....,k\}$ whose T_j is the maximum is the index of the maximum probability. Once again, the value of the maximum probability is defined by $\frac{T_j}{2^{P_{d,\min}}}$ or $T_j >> 2^{P_{d,\min}}$. It is obvious that the algorithm does not completely require finding the exact value of the probability. In addition, it does not take any division operation at all. In addition, if only the index of the maximum probability is required, even the right shift operation is not necessary to perform.

IV. A Sample Application of the Proposed Algorithm: A HMM-Based Motion Type Classifier

In this section, the sample implementation of the proposed forward algorithm on an embedded system platform is described. The results, in terms of classification correctness, code size, memory requirement and execution time, are compared with the ordinary and the scaling coefficients forward algorithms.

A. Acceleration Data Capturing

In our sample application, the ADXL346 development board [8] was used as an acceleration data-logger. It was programmed to capture the data at 100Hz sampling rate with the maximum acceleration of 4G. During the data collection stage, the data-logger was firmly attached to the middle of the upper leg of the subject. Then, the subject was requested to repeatedly perform the same type of motion for a period of 5 minutes (total number of data are $5 \times 60 \times 100 = 30{,}000$). The captured motion types consisted of:

1) Cycling
2) Jogging
3) Sit-stand-sit
4) Motionless (i.e. Sitting or Standing)
5) Walking
6) Walking upstairs and downstairs
7) Slow walking

From this point on, the number preceding these motion types are also used to refer to the motion type.

After obtaining the acceleration data for each motion type, the following steps were performed on a motion type by motion type basis to preprocess the data and make it ready for HMM construction:

1) Transferred the acceleration data file from the MicroSD memory card of the ADXL346 to a personal computer. The acceleration data were stored to the file in the following format: index, hour, minute, second, second/128, Da_x, Da_y and Da_z. The first 5 fields are the time stamp. The last 3 fields are the acceleration data sampled from the embedded acceleration sensor of the ADXL346.
2) Replaced the time stamp with the more accurate delta time between a pair of sampled data, converted all sampled data from (Da_x, Da_y, Da_z) to (A_x, A_y, A_z) by multiplying each field with 0.0039 and calculated the magnitude of acceleration data by using: $A = \sqrt{A_x^2 + A_y^2 + A_z^2}$.
3) Selected only part of the acceleration data which were relevant to the captured motion type and kept only the continuous 3 minutes of acceleration data to the output file.

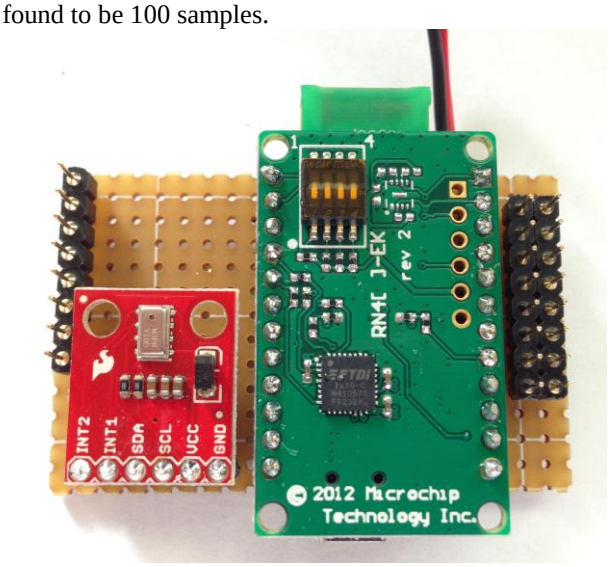
B. HMMs Construction

The Matlab scripts were coded to facilitate HMMs construction to produce a set of HMM parameters: $\{A, B, \Pi\}_i$, where $i \in \{1,2,....,7\}$ (there are 7 motion types as detailed in Sec. IV-A). Several experiments confirmed that the motion type classification correctness was the highest when HMMs were constructed by use of the acceleration data right away. Experiments were iteratively performed to find the set of HMM parameters and the best motion type classification results. The set chosen for implementation on the embedded system platform (details given in Sec.IV-C) must have small dimensions of both A and B. As the dimension of A is directly proportional to the required operations while the dimension of B is directly proportional to the memory required to store the emission probabilities. In our implementation, the dimensions of A and B were chosen to be 2 and 16, respectively. This gave rise to the average verification result of motion type classification correctness of 79.9 % on a personal computer. Although the chosen configuration of HMM parameters for our implementation is fairly far from perfect in term of classification correctness, it is a good starting point to test the proposed algorithm. More configurations that could give rise to the higher motion type classification correctness are being pursued.

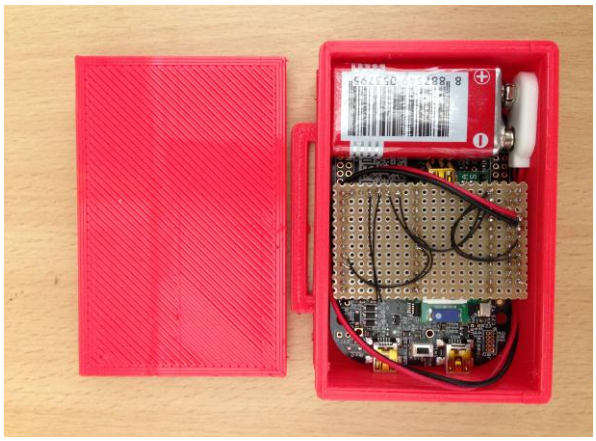
C. HMMs Verification

During the data capturing stage, the subject was also requested to perform the following sequence of motion types: motionless, sit-stand-sit, walking, jogging, cycling, walking upstairs and downstairs and slow walking. Each motion type was performed for a period of 1 minute. The captured acceleration data file was used to verify the motion type classification correctness. Experiments were performed on a personal computer by use of Matlab scripts to find the number of samples of data, which is equivalent

to the dimension of the observations O, which gave rise to the highest motion type classification correctness. This was found to be 100 samples.



(a)



(b)

Figure 1. The prototype of the embedded HMM based motion type classification platform: (a) The add-on Bluetooth module and (b) The overall embedded system.

D. Implementations

In order to compare the benefits of the proposed forward algorithm against the previously proposed ones, it was implemented on an embedded platform. The ordinary and the scaling coefficients forward algorithms were also coded on the same platform. The mBed compiler (www.mbed.org) was chosen as the main development tool. The target platform for all implementations was the Freescale KL25Z. The platform is an ultra-low-cost development platform for Kinetis L Series KL1x (KL14/15) and KL2x (KL24/25) MCUs built on ARM Cortex-M0+ processor. It embeds the MKL25Z128VLK4 MCU running at 48MHz with 128KB flash and 16KB SRAM. In addition to the platform, the Bluetooth module was interfaced to transmit the classification result to a personal computer. The prototype of the embedded platform is shown in Fig. 1. Followings were the details of the implementation:

- Use a set of HMM parameters: $\{A, B, \Pi\}_i$ where $i \in \{1,2,....,7\}$ previously trained on a personal computer by use of the Matlab scripts whose dimensions of A and B were 2 and 16, respectively.
- Employ the state machine approach for processing all the forward algorithms. The timer interrupt is used to periodically capture the acceleration data at a period of 10 ms (100 Hz). Once data is captured, the forward algorithm is invoked to operate on the captured data, o_k, in a k-state. As we have a set of 7 HMMs (7 motion types), this means that the operations must be applied to all HMMs in a single state. After processing 100 samples of data, or 100 states, all 7 probabilities are obtained, the maximum probability is resolved, and the motion type is determined.
- Verification data were prepared to be in the form of a string consisting of a continuous sequence of characters. Each character represents the acceleration data whose value is in the range of $[0,15]$. The verification data were transferred via a USB port to the platform instead of relying on the platform's embedded acceleration sensor. Doing this makes it easy for us to verify the motion type classification correctness between different implementations.
- Measure the execution time taken in all states by use of the mBed's built-in class: Timer. Also measure the code size and static memory requirement by use of the mBed's program details.

Followings are an algorithm specific implementation:

- The ordinary and the scaling coefficients forward algorithms represent all data by use of double. It is noted that the float data type was previously employed in testing the implementation, it came out that the algorithms could not perform correctly when the number of state, which is equivalent to the sequence of acceleration data, is greater than 50.
- The proposed forward algorithm represents all data by use of unsigned integer. The constants are chosen to be: $A_f = B_f = 1024, \Pi_f = 32$ and $D = BN = 65{,}536$. These constants were chosen as they gave rise to the values of P_n and P_d which were capable of handling by the embedded system platform and, at the same time, did not cause arithmetic overflow.

With respect to the verification data, all implementations give rise to 100 % motion type classification correctness compared to the classification result running in offline mode by the matlab scripts on a personal computer. Fig. 2 illustrates: (top) the acceleration data waveform of different motion types and (bottom) the motion type classification results where the numbers on the y-axis represents the motion type; i.e. 1: Cycling, 2: Jogging, 3: Sit-stand-sit, 4: Motionless (Sitting).

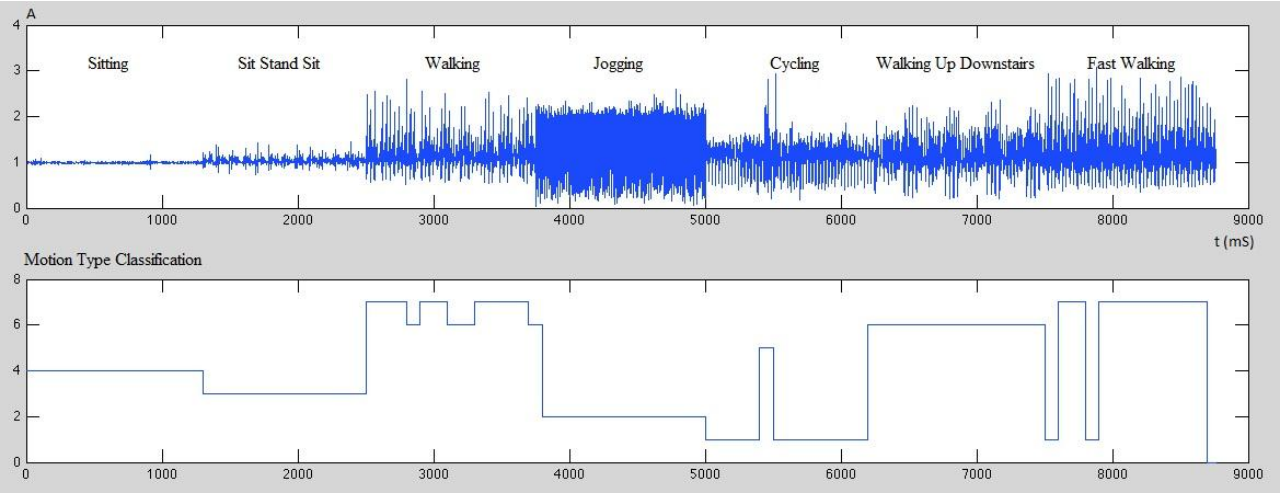


Figure 2. The acceleration data waveform and the motion type classification results.

E. Experimental Results and Discussions

In order to proof the advantages of the proposed algorithm, the execution times in the three main states of the algorithm; the initialization (T_{init}), the induction (T_{ind}) and the termination (T_{term}) states, are measured and averaged during the verification. Table I presents the comparison of the execution times of all implementations. It is obvious that our proposed algorithm outperforms the ordinary and the scaling coefficients ones in all states. The scaling coefficients forward algorithm that was proposed to keep the probability values in the dynamic range of the machine is the slowest in term of execution times. This comes from the fact that it takes time to scale down the coefficients during the initialization and induction states. In addition, the algorithm spends time during the termination state in order to calculate the probabilities by use of computationally expensive operations of logarithmic and exponential. The ordinary forward algorithm performs better than the scaling coefficients one with faster execution times in all states. But it must be taken into account that our implementation makes use of the data representation which is capable to process a sequence of data without underflow. If the sequence of data is longer than 100 samples, it might cause the underflow problem and results in incorrect classification result. It is also possible to speed up the execution by reducing the size of the sequence of data to be less than 50 and make use of float data type in the implementation of the ordinary algorithm. However, the proposed algorithm still outperforms the ordinary one as it operates only on an integer data type.

From the execution time in the three main states, it can be implied that the proposed algorithm forces the processor to sleep the longest. With the state machine approach of implementation driven by timer interrupt at a period of 10 ms and the sequence of data of size 100, during one second the total execution time, defined by $T_{init} + 98 \times T_{ind} + T_{term}$, of our proposed algorithm is only 12 + 98 × 30 + 20 = 3.082ms. While the ordinary and the scaling coefficients forward algorithms take 107 + 98 × 360 + 45 = 35.432ms and 316 + 98 × 570 + 44049 = 100.225ms, respectively. Therefore, it can be concluded that our proposed algorithm is likely to consume the least power consumption.

In addition to the execution time, the code size and the static memory requirement of all implementations are also considered. The results are shown in Table II. It can be seen that as a result of employing the integer representation and integer only operators the code size (Flash) and the RAM usage of our proposed forward algorithm are the minimum.

TABLE I. COMPARISON OF THE EXECUTION TIMES IN THE THREE MAIN STATES OF ALL IMPLEMENTATIONS

State	Execution Time (us)		
	Ordinary Algorithm	Scaling Coefficients	Our Proposed One
T_{init}	107	316	12
T_{ind}	360	570	30
T_{term}	45	44,049	20

TABLE II. COMPARISON OF THE CODE SIZE AND THE STATIC MEMORY REQUIREMENT OF ALL IMPLEMENTATIONS

Memory Type	Memory Requirement (kB)		
	Ordinary Algorithm	Scaling Coefficients	Our Proposed One
Flash	19.6	23.3	17.8
RAM	8.1	10.9	4.4

V. CONCLUSION

The hidden Markov model has a set of hidden states, Q, the output observations, O, transition probabilities, A, output (emission) probabilities, B, and initial state probabilities, Π. Given the hidden Markov model parameters $\{A, B, \Pi\}$, the forward algorithm computes the probability of a particular output sequence O. The drawback of the algorithm occurs when performing on a long output sequence, the extremely small probability values causes underflow on most machines. The most common solution to this problem is to use scaling

coefficients that keep the probability values in the dynamic range of the machine, and that are independent of the size of the observation sequence. In this paper, the forward algorithm targeted for embedded platform implementation is proposed. In contrast to the previously proposed algorithms that rely on representing and operating on floating point data type (float or double), our algorithm uses all integer representation and operations. The sample application of the proposed algorithm is described to be used for embedded motion type classification; 7 motion types with 2×2 transition probabilities and 16 emission probabilities, whose processor is a 32-bit ARM Cortex-M0+. The classification results are proved to be comparable to the ordinary and the scaling coefficients forward algorithms. The benefits of the proposed algorithm are three folds which are: reduction of the execution times, code size, and memory requirement. Additionally, the algorithm seems to force the processor to sleep the longest. That is to say its total execution time is only 3.082ms per second while the ordinary and the scaling coefficients forward algorithm take 35.432 and 100.224ms, respectively. This confirms that the proposed algorithm preserves the overall power of the target embedded system platform.

ACKNOWLEDGMENT

This work was funded by Walailak University under project: Development of a Hidden Markov Based System for Motion Types Recognition and Classification from Body Attachment Sensors and Optimization of a Hidden Markov Model Based Fall Detection Algorithm for Embedded System Platform. The author would like to thank ARM Ltd., Nuvoton Technology Corp., Freescale Semiconductors Corp., Microchip Technology Inc. and Analog Devices Corp. for providing us the embedded system platforms.

REFERENCES

[1] (Dec. 2012). Hidden markov model. *Wikipedia - The Free Encyclopedia* [Online]. Available: http://en.wikipedia.org/wiki/Hidden_Markov_model

[2] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[3] C. C. Yang and Y. L. Hsu, "A review of accelerometry-based wearable motion detectors for physical activity monitoring," *Sensors*, vol. 10, no. 8, pp. 7772-7788, 2010.

[4] J. Li, S. Chen, and Y. Li, "The fast evaluation of hidden Markov models on GPU," in *Proc. IEEE International Conference on Intelligent Computing and Intelligent Systems*, Nov. 2009, pp. 426-430.

[5] S-Z Yu and K. Hisashi, "Practical implementation of an efficient forward-backward algorithm for an explicit-duration hidden Markov model," *IEEE Transactions on Signal Processing*, vol. 54, no. 5, pp. 1947-1951, May 2006.

[6] J. Li, Y. Li, and S. Chen. "The parallel evaluation of hidden Markov models on graphic processing units in supervised recognition," in *Proc. 2nd International Conference on Computer Engineering and Technology (ICCET)*, Apr. 2010, pp. 135-139.

[7] S. J. Melnikoff, S. F. Quigley, and M. J. Russell, "Speech recognition on an FPGA using discrete and continuous hidden markov models," in *Proc. International Conference on Field Programmable Logic and Applications*, 2002, pp. 202-211

[8] Analog Devices Inc. (2014). ADXL345 datalogger/development board. [Online]. Available: http://www.analog.com/en/mems-sensors/mems-inertial-sensors/adxl345/products/EVAL-ADXL345Z-DB/eb.html

W. Kurdthongmee received his Ph.D. in Computer Science (Computer Graphics and Solid Modeling) from Brunel University, UK, in 1998. He received his B.Sc. and M.Sc. in Physics from Prince of Songkhla University, Hatyai Campus, Thailand in 1987 and 1994, respectively. Currently, he is an associate professor (computer engineering) in School of Engineering and Resources Management, Walailak University, Thailand. His research interests are related to real-time embedded systems, FPGA-based systems, system-on-chip, neural networks, machine visions and image processing.

