Hidden Markov Models Implementation for Tangible Interfaces

Piero Zappi[1], Elisabetta Farella[1], and Luca Benini[1]

DEIS University of Bologna, Bologna, Italy
{piero.zappi,elisabetta.farella,luca.benini}@unibo.it

Abstract. Smart objects equipped with inertial sensors can recognize gestures and act as tangible interfaces to interact with smart environments. Hidden Markov Models (HMM) are a powerful tool for gesture recognition. Gesture recognition with HMM is performed using the forward algorithm. In this paper we evaluate the fixed point implementation of the forward algorithm for HMM to assess if this implementation can be effective on resource constraint devices such as the Smart Micrel Cube (SMCube). The SMCube is a tangible interfacet that embeds an 8-bit microcontroller running at 7.372 MHz. The complexity-performance trade off has been explored, and a discussion on the critical steps of the algorithm implementation is presented.

Keywords: Smart Object, Hidden Markov Models, Tangible interfaces, Fixed point.

1 Introduction

The development of smart objects is an active field of research. With the objective of enhancing the interaction with smart environments, smart objects can be used as tangible interfaces and play a fundamental role in improving human experience within interactive spaces for entertainment and education [1]. The SMCube is a tangible interface developed for the *TANGerINE* framework, a tangible tabletop environment where users manipulate smart objects in order to perform actions on the contents of a *digital media table* [2]. Previous work showed how a simple ad-hoc technique and a standard tree classification algorithm can be implemented on the SMCube to include basic gesture recognition and improve interaction modalities [3].

Hidden Markov Models (HMMs) allow to handle temporal dynamics and classify more complex gestures than the ones already recognized. Typically, classification with HMMs is performed using a recursive algorithm called *forward* algorithm. Although this process is a lightweight task, several issues must be considered in order to implement it on a low-power, low-cost microcontroller such as the one embedded on the SMCube.

In this paper we evaluate the fixed point implementation of the forward algorithm. Discussion of ad-hoc solution to solve numerical problems while keeping low overall computational complexity is presented. Considerations about the complexity of the algorithm, both in terms of computational and memory cost, and its performance are discussed through the paper.

2 Hardware Overview

The SMCube is a smart object equipped with sensors (a digital tri-axes accelerometer from STM, LIS3LV02DQ, and 6 photo transistors) and actuators (infrared LEDs) (see figure 1). It embeds an ATMega 168 low-power, low-cost microcontroller to sample and process data from its sensors, and a Bluetooth 2.0 transceiver from BlueGiga (WT12) to wirelessly communicate with a PC. The ATMega 168 features a RISC architecture that can operate up to 24MHz and offers 16 KB of Flash memory, 1 KB of RAM and 512 Bytes of EEPROM. The microcontroller includes a multiplier and several peripherals (ADC, timers, SPI and UART serial interfaces etc.). The firmware has been implemented in C using the Atmel AVR Studio 4 IDE that provides all the APIs necessary to exploit the peripherals and perform operations with 8, 16, and 32 bit variables.

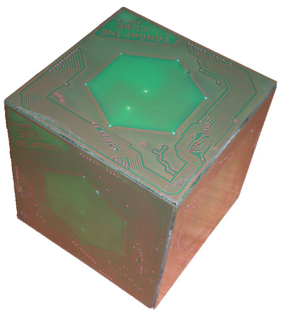

Fig. 1. The Tangerine SMCube. The cube edge is 6.5 *cm* long.

3 Hidden Markov Models

A HMM is a probabilistic model used to describe sequences of observations $O = \{o_1, o_2, ..., o_T\}$ and their corresponding hidden state $Q = \{q_1, q_2, ..., q_T\}$.

A discrete HMM is characterized by a set of N states (s_i), M possible observable (measurable) values (v_k), a $N \times N$ matrix state transition $A = \{a_{ij}\} = P(q_{t+1} = s_j | q_t = s_i)$, a $N \times M$ observation matrix $B = \{b_i(k)\} = P(o_t = v_k | q_t = s_i)$, and a $N \times 1$ starting probability vector $\Pi = \{\pi_i\} = P(q_1 = s_i)$. The compact notation for an HMM is $\lambda = (A, B, \Pi)$.

Two hypotheses are given: (1) the state at time t depends only on the state a $t-1$, $P(q_t|q_{t-1})$; (2) each observation is independent given its state, $P(o_t|q_t)$.

There are three problems associated with HMM: (1) the classification problem, (2) the decoding problem, (3) the training problem [4].

Given a model λ and a sequence of observations O, the classification problem consists in finding the probability that λ generated O and is solved using the *forward* algorithm. When we use HMMs for gesture classification, we train one model, λ_i for each class of gestures, then we apply the forward algorithm with all the models and find the most probable one, $G_{out} = argmax_i[P(O|\lambda_i)]$.

3.1 The Forward Algorithm

The forward algorithm is a recursive algorithm that relies on a set of support variables $\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = i|\lambda)$ and is made up of three steps.

1. Initialization: $\alpha_1(i) = \pi_i(O_1)b_i(O_1)$, $1 \leq i \leq N$
2. Induction: $\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i)a_{ij}]b_j(O_{t+1})$, $1 \leq j \leq N$ and $1 \leq t \leq T-1$
3. Termination: $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

3.2 Normalization

According to the definitions in the previous section we can see that the $\alpha_t(j)$ are sum of a large number of terms in the form $(\prod_{s=1}^{t-1} a_{q_s,q_{s+1}} \prod_{s=1}^{t} b_{q_s}(O_s))$. Since both the a_{ij} and the $b_i(k)$ are smaller than 1 as t become large $\alpha_t(j)$ tends to zero exponentially and soon it exceedes the precision of any machine.

In order to avoid underflow, the $\alpha_t(j)$ are normalized at every step using the scaling factor $c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}$. The scaled $\widehat{\alpha}_t(j)$ are used in place of the $\alpha_t(j)$.

This normalization procedure is not suitable for low-power microcontrollers since it requires to perform N divisions each time a new sample is processed.

Thus we propose an alternative approach:

1. check if all $\alpha_t(j)$ are smaller than $\frac{1}{2}$, otherwise scaling is not needed;
2. calculate the number of shift to the left (l) needed to render the highest $\alpha_t(j)$ greater than $\frac{1}{2}$;
3. shift all $\alpha_t(j)$ to the left of l bits.

This procedure requires only shifts and can be efficiently implemented on a microcontroller.

3.3 Likelihood

To compute the final sequence probability we can not use the scaled $\widehat{\alpha}_t(j)$. However we can notice that:

$$\sum_{i=1}^{N} \widehat{\alpha}_T(i) = \prod_{t=1}^{T} 2^{l_t} \cdot \sum_{i=1}^{N} \alpha_T(i) = \prod_{t=1}^{T} 2^{l_t} \cdot P(O|\lambda) = r \longrightarrow P(O|\lambda) = \frac{r}{\prod_{t=1}^{T} 2^{l_t}} \quad (1)$$

Since $P(O|\lambda)$ can be very small, we compute $\log P(O|\lambda) = \log(r) - \sum_{t=1}^{T} \log 2^{l_t}$.

If we decide to use $\log_2$ we already have the value of $\sum_{t=1}^{T} \log 2^{l_t}$ by keeping track of how many shifts we performed for scaling. Furthermore, we do not need to compute $\log(r)$ since logarithm is a monotonically increasing function. Thus, to compare 2 models, we simply check for the one that required less shifts for scaling, in case of tie the one with higher r is the most probable model.

4 Evaluation

To evaluate this implementation we used a dataset made up of 10 complex gestures collected on the car assembly scenario [6]. These gestures can be compared to the ones that may be used within role-playing games. The dataset has been extended since its first use and now it includes 70 repetition for each gesture. We used 4 fold cross validation technique to extend the validation set up to all 70 samples.

To recognize these gestures we used a set of discrete HMMs with 4 states ($N = 4$). Accelerometer' streams have been quantized to 3 symbols ($M = 3$).

To assess the complexity of the forward algorithm we assumed the values presented in tables 1a and 1b, where N is the number of HMM states and C is the number of gestures we want to recognize (here $C = 10$). The memory cost is given by $\frac{\text{data size}}{8} \cdot C \cdot (N^2 + N \cdot M + 2 \cdot N)$. Where M is the number of symbols in the accelerometer stream.

4.1 Classification Accuracy

To evaluate the performance loss due to the use of fixed point data representation, we classified the dataset using a floating point representation of the data and the traditional normalization algorithm (optimal performance), and using a fixed point representation and the shift scaling algorithm.

Table 1. Computational complexity

(a) Computational complexity

Operation	Cost
Shift	1
Variables comparison	1
Sum 8 bits	1
Sum 16 bits	2
Sum 32 bits	4
Multiplication 8 bits	2
Multiplication 16 bits	4
Multiplication 32 bits	6

(b) Algorithm complexity

Algorithm	Cost
$\alpha_{t+1}(i)$ Calculation	$(N+1)$ mul. $+ N$ sum.
Normalization	$2 \cdot N + 1 + 2 \cdot$ data size
Single step (8-bit)	$C \cdot [N \cdot (3 \cdot N + 2) + 2 \cdot N + 17]$
Single step (16-bit)	$C \cdot [N \cdot (6 \cdot N + 4) + 2 \cdot N + 33]$
Single step (32-bit)	$C \cdot [N \cdot (10 \cdot N + 6) + 2 \cdot N + 65]$

Performances are evaluated using the following indexes (see table 2):

- *Correct Classification Ratio*: $CCR = \frac{\text{number of correctly classified instances}}{\text{total number of instances}}$; is a global indication of the performance of the classifier.
- *Precision*: $PR_i = \frac{\text{number of instances correctly classified for class i}}{\text{number of instances classified as class i}}$; is an indication of the exactness of the classifier.
- *Recall*: $RC_i = \frac{\text{number of instances correctly classified for class i}}{\text{total number of instances from class i}}$; is an indication of the performances of the classifier over a specific class

5 Discussion

Table 2 and 3 present PR, RC, CCR, computational and memory cost for our implementations when using the given dataset. The implementations that use 16 and 32 bits fixed point data representation achieve similar or even equal CCR than the floating point solution. On the other hand the 8 bits fixed point implementation worsen the CCR by 14.15 %. However, the 32 bit solution can not be implemented on the ATmega168 since it requires more RAM than available, therefore the 16 bits solution is the optimal choice for the SMCube.

Table 2. Classification performances

Class	*PR* 8b	*PR* 16b	*PR* 32b	*PR* fl	*RC* 8b	*RC* 16b	*RC* 32b	*RC* fl
Gesture 1	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.99
Gesture 2	0.50	0.66	0.66	0.66	0.01	0.64	0.64	0.64
Gesture 3	0.38	0.54	0.54	0.54	0.41	0.56	0.56	0.56
Gesture 4	0.54	0.60	0.61	0.61	0.64	0.67	0.69	0.69
Gesture 5	0.29	0.67	0.69	0.69	0.36	0.50	0.50	0.50
Gesture 6	0.36	0.53	0.53	0.53	0.43	0.36	0.36	0.36
Gesture 7	0.53	0.65	0.65	0.65	0.59	0.86	0.86	0.86
Gesture 8	0.47	0.56	0.56	0.56	0.52	0.63	0.63	0.63
Gesture 9	0.77	0.87	0.87	0.87	0.81	0.89	0.89	0.89
Gesture 10	0.93	0.96	0.96	0.96	0.90	0.99	0.99	0.99
CCR					**56.71%**	**70.71%**	**70.86%**	**70.86%**

Table 3. Performance and cost comparison

Variables Size (bits)	**CCR (%)**	**Memory cost (bytes)**	**Computational cost**
8	56.71	360	810
16	70.71	720	1370
32	70.86	1440	2090
Floating point	70.86		

6 Conclusion and Future Works

In this paper we presented our evaluation of a fixed point implementation of the forward algorithm for HMM. Furthermore, we presented our solutions to the peculiar numerical problems of this classification algorithm. The 16-bit implementation is the best solution that can be implemented on our target microcontroller (ATMega168). This solution shows performance only slightly worse than the optimal ones of the floating point implementation (70.71% CCR, 16 bit fixed point; 70.68% floating point) and makes this implementation suitable for smart objects equipped with low-power, low-cost microcontrollers such as the SMCube.

HMM is a common approach in gesture recognition, thus the possibility to implement this algorithm on a smart object greatly enhances potential for using it as an effective HCI device. In future works we plan to augment human computer interaction within a tabletop environment allowing the interaction through a set of complex natural gestures.

References

1. Ishii, H.: The tangible user interface and its evolution. J. Comm. ACM 51(6), 32–36 (2008)
2. Baraldi, S., Del Bimbo, A., Landucci, L., Torpei, N., Cafini, O., Farella, E., Pieracci, A., Benini, L.: Introducing tangerine: a tangible interactive natural environment. In: Proc. of ACM International Conference on Multimedia (MM), pp. 831–834. ACM Press, Augsburg (2007)
3. Cafini, O., Farella, E., Benini, L., Baraldi, S., Torpei, N., Landucci, L., Del Bimbo, A.: Tangerine SMCube: a smart device for human computer interaction. In: Proc. of IEEE European Conference on Smart Sensing and Context (2008)
4. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
5. Mitra, S., Acharya, T.: Gesture Recognition: A Survey. IEEE Transactions on Systems, Man and Cybernetics - Part C 37(3), 311–324 (2007)
6. Zappi, P., Stiefmeier, T., Farella, E., Roggen, D., Benini, L., Tröster, G.: Activity Recognition From On-Body Sensors by Classifier Fusion: Sensor Scalability and Robustness. In: Proc. 3rd Int. Conf. on Intelligent Sensors, Sensor Networks, and Information Processing (2007)