Recovering binary data transmitted over unknown communication channels
A DSP implementation

C. Léal [(1)], *C. Meilhac* [(1)], *A. Pesme* [(1)], *J.-F. Bercher* [(2)] *and C. Vignat* [(3)],

[(1)] Engineer Students at ESIEE
e-mail: {lealc, meilhace, pesmea}@esiee.fr
[(2)] Équipe Communications Numériques Sans Fil, LPSI, ESIEE
e-mail: bercherj@esiee.fr
[(3)] Laboratoire Systèmes de Communication, Université Marne-la-Vallée
e-mail: vignat@univ-mlv.fr

ABSTRACT:

The problem of recovering source data transmitted over an unknown or partially known channel from the sole observation of the received data is a major challenge in the telecommunication field since last decade. The objective of this paper is twofold: first, we exhibit a new approach for this task, based on a simple Markov model that governs the received data in the case of binary inputs. Second we describe the implementation of this approach on a TMS320C62 DSP, and show that this method could raise interest in an industrial context. All these points are illustrated through simulations.

1. INTRODUCTION

We consider the usual model of a communication system where the emitted data consist of independent binary symbols. They are transmitted through a linear channel that involves:

- the dispersion effect induced by propagation,
- the multipath phenomenon (that appears for instance in mobile communication systems).

The transmission channel is commonly modeled by a finite impulse response filter; moreover, the output is usually corrupted by Gaussian white noise that represents observation noise and interfering signals. In this context, qualified as "blind context", the channel is generally fully unknown, as well as, obviously, the emitted data. The problem consists thus in recovering the emitted data, from the sole observation of the received symbols.

The key point of our approach is to remark that the noiseless output data take discrete values in a finite set of dimension 2^p (where p is the length of the impulse response). The output of the channel evolves as a Markov chain. The transition probabilities are known, but the observations are unassigned to their respective states. In the noisy case, the model is a Hidden Markov Model (HMM).

Thus, identification of the Markov parameters will allow to recover the binary input sequence, and also give, as a by-product (although unnecessary here), the explicit value of the channel, up to a sign indeterminacy.

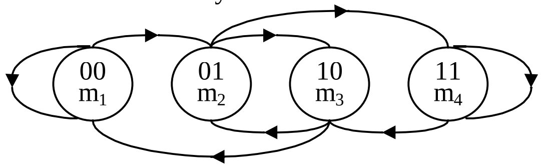
We had to derive a specific algorithm, which looks like a Viterbi algorithm, which is not usable in the considered situation because the channel is unknown.

In the noisy case, we developed a learning process of the Hidden Markov Model. This process provides a set of solutions, each one being characterized by its likelihood. A final detection step has then to be performed.

Equipment used:
The project was realized using the following equipment: a PC, an evaluation board for the TMS320C62 with the associated software (assembler, linker, debugger), C language tools and the numerical computation software Matlab™.

2. PRINCIPLES OF THE METHOD

We start from the remark that the noisy output data is the observation sequence of a hidden Markov model (HMM). Each state of the HMM is simply the sequence defined by the p last emitted bits. Thus, there are only 2 possible transitions from state at time t to the state at time t+1, corresponding to the emission of one bit by the source.



This work was performed at ESIEE by three engineer students during a 6 weeks training period, under supervision of JFB and CV.

The previous figure shows a HMM representation for a simple 2 coefficients transmission channel.

The HMM $\Lambda = (A,B,\pi)$ is characterized by:

- N, the number of states in the model ($N=2^p$ for our application). We denote the individual states as $S=\{S_1, S_2,..., S_N\}$, and the state at time t as vector q_t. S is associated with $V=\{m_1, m_2, ... m_N\}$, where m_i represents the noiseless observation output for state S_i.
- the state transition probability matrix $A=\{a_{ij}\}$, where $\forall\ 1 \le i, j \le N \quad a_{ij} = P[q_{t+1} = S_j \mid q_t = S_i]$. The a_{ij} are the transition probabilities from state i to state j. In our special case, any state at time *(t+1)* can be reached from 2 states at time *(t)* because the source data are supposed binary distributed. Thus, in the case of equiprobable bits, a_{ij} = 1/2 if there is a transition from state S_i to state S_j ; and 0 otherwise.
- the law of the noisy output in state S_i, $B=\{b_i(O_t)\}$. A common hypothesis is that the corrupting noise is Gaussian $N(0,\sigma^2)$, so that $b_i(O_t)$ can be written as a Gaussian law $N(m_i,\sigma^2)$:
$$b_i(O_t) = \frac{1}{\sqrt{2\cdot\pi\cdot\sigma^2}}\exp\left(-\frac{(O_t - m_i)^2}{2\cdot\sigma^2}\right),$$
where m_i is the mean of the observation symbol associated to the state S_i, and σ^2 the variance of the observation, for any state.
- the initial state distribution $\pi=\{\pi_i\}$ where $\pi_i = P[q_1=S_i]$, $1 \le i \le N$.
Without prior knowledge, a non-committal choice (*la raison insuffisante* of Laplace) is to take all states equiprobable:
$$\pi = \left[\frac{1}{N};\frac{1}{N};...;\frac{1}{N}\right].$$

Given $O = [O_1\ O_2\ ...\ O_T]$ and the HMM model $\Lambda = (A, B, \pi)$, we face the two classical problems in HMM modeling:

- ➢ how to choose a state sequence Q = $[q_1\ q_2\ ...\ q_T]$ which best explains the observations,
- ➢ how to learn and adjust the model parameters $\Lambda = (A, B, \pi)$ in order to maximize $P(O \mid \Lambda)$.

Solution to the first problem:
In order to solve the first problem, we use an algorithm based on the Viterbi algorithm. To find the single best state sequence $Q=\{q_1\ q_2\ ...\ q_T\}$ given observation sequence $O = \{O_1\ O_2\ ...\ O_T\}$, we define the probability:
$$\delta_t(i) = \max_{q1,q2,...qt-1} P[q_1\ q_2\ ...\ q_t=i, O_1\ O_2\ ...\ O_t \mid \Lambda]$$
i.e., $\delta_t(i)$ is the probability of the most likely path among all paths that end in state S_i at time t and generate the observation sequence $\{O_1\ O_2\ ...\ O_T\}$. By induction, we have:
$$\delta_{t+1}(j) = \left[\max_i \delta_t(i)\cdot a_{ij}\right]\cdot b_j(O_{t+1}) \qquad (1)$$
To retrieve the state sequence, we need to keep track, for each t and state S_j, of the argument that maximizes (1). Let us denote by $\Psi_t(j)$ the array that records these arguments.

Then, the complete procedure for finding the best state sequence is:

1) Initialization step:
$\delta_t(i) = \pi_i \cdot b_i(O_1)$, $1 \le i \le N$
$\Psi_t(i) = 0$.

2) Recursion step: $\forall\ 2 \le t \le T$, $\forall\ 1 \le j \le N$
$\delta_t(j) = \max_{1\le i\le N}\left[\delta_{t-1}(i)\cdot a_{ij}\right]\cdot b_j(O_t)$,
$\Psi_t(j) = \arg\max_{1\le i\le N}\left[\delta_{t-1}(i)\cdot a_{ij}\right]$.

3) Termination step:
$P^* = \max_{1\le i\le N}\left[\delta_t(i)\right]$,
$q_T^* = \arg\max_{1\le i\le N}\left[\delta_t(i)\right]$.

4) State sequence recovering:
$q_t^* = \Psi_{t+1}(q_{t+1}^*)$, t = T-1, T-2,..., 1

With this modified Viterbi algorithm, we do not have to identify the unknown channel in order to recover the emitted data (*i.e.* the state sequence). Identification of the HMM leads directly to the emitted data associated with the observation sequence. As a matter of fact, information about the channel impulse response is in the set of means $\{m_i\}$, which have to be initialized, and as described in the following, these first estimates are then refined in line during the reconstruction process.

Solution to the second problem:
The previous procedure need an initial HMM model of the channel. There are two ways to build such initial model:

① sending , and using, a known binary sequence in order to estimate the unknown parameters (m_i, σ^2), ② designing a learning process so as to estimate the transition matrix A, together with the set of means.

① Training sequence:
We designed minimum-length training sequences using a pseudo-noise generator. This generator consists in a shift register of length *p* looped with XOR. It allows producing 2^{p}-1 states of the model without redundancy. The last state, a sequence of *p* zeros, is added “by hand”. The training sequence is of length is $p2^{p}$, which is the minimum length possible.

② Non supervised learning of the model:
In a first step, a set of empirical means is estimated from the observation sequence (the exact procedure will be described elsewhere). Then, by recensing transitions from one mean to another, we obtain an estimate of transition matrix *A*. This empirical transition matrix is then compared to the expected matrix, and by rearrangements, one can assign means to states. Thus the model is initialized and the algorithm for the recovery of the binary input can be engaged on the whole sequence. This approach enables the algorithm to operate in blind context.

The first model has to be refined synchronously with the observation sequence. Thus, at each time *t*, we adjust the HMM parameters through the m_i and σ^2, in order to maximize the probability of the observation sequence given the model. This task is achieved using principles analog to those of the Baum/Welch algorithm.
We introduced the following weights:

$$\alpha_i(t) = \frac{\delta_t(i)}{\sum_{k=1}^{t} \lambda^{t-k} \cdot \delta_k(i)}, \ \lambda < 1$$

which take observations into account proportionally to the likelihood that state *S*=*Si* at time *t*. We also introduced here a forgetting factor λ which enables to forget the old parameters and thus provides a tracking capability in a non stationary environment. This leads to the following reestimation formula:

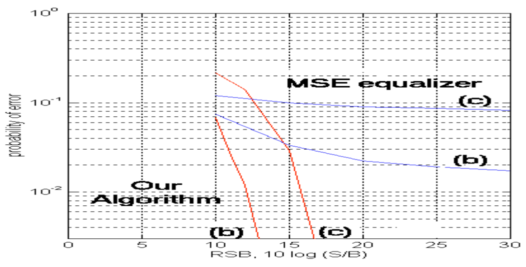
$$\forall \ 1 \le i \le N \ , \ \forall \ 2 \le t \le T :$$
$$m_i(t) = (1-\alpha_i(t)) \cdot m_i(t-1) + \alpha_i(t) \cdot O_t \ ,$$
$$\mathrm{var}(t) = (1-\alpha_0) \cdot \mathrm{var}(t-1) + \alpha_0 \cdot (O_t - m_0)^2 \ ,$$

where α_0 and m_0 are associated with parameters with most likely state at time *t*, given observation O_t.

3. SIMULATION RESULTS

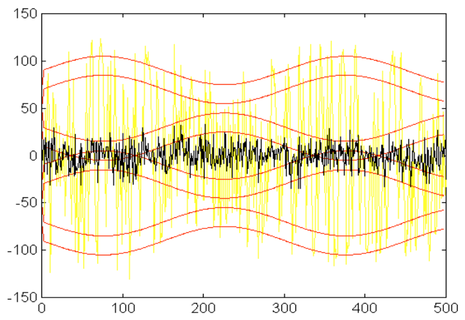
Monte-Carlo simulations were performed under MATLAB. Tests were repeated 100 times on observation sequence of length 500, with a forgetting factor set to 0.99995. Three kinds of transmission channels were analyzed: a) well-known test cases, b) non-stationary channels, and c) channels with echo.

a) We tested the three transmission channels (a), (b) and (c) from the book of Proakis [6]. These 3 channels are often used as test cases for evaluation and comparison of algorithms.
Results are reported in the following figure, which gives the error rate performance of a linear MSE equalizer (31 coefficients), versus the algorithm studied here:



The 2 methods are almost identically efficient for low RSB applications; but our algorithm becomes more efficient as the RSB increases, and is faster than the MSE equalizer method.
However, our algorithm was not tested with channel (a), which, with 11 coefficients, requires too much memory resources.

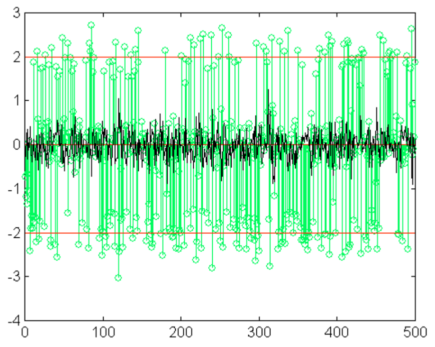
b) Non stationary channels
To perform the simulation of non-stationary channels, we used a 3 coefficients channel:
[10 X 50], with $X = 30 + 15 \cdot \sin(2\pi \ 0.01t)$.



The previous figure shows the varying noiseless output of the channel superposed to the effective observations and the noise:

The results of simulations confirm once again the efficiency of the algorithm which succeeds to track the evolution of the channel and recovers the input sequence without any error up to a signal to noise ratio of 13 dB.

c) Transmission channels with echo:
For this experimentation, we used a 6 coefficients channel [1 0 0 0 0 1]. The following figure shows the levels of the noiseless output of this channel superposed to the effective observations and noise:



The main problem encountered for such channels is the low number of noiseless output (3) in comparison with the number of states of the HMM (64). This means that many states have the same noiseless output value. But once again, tests proved that the algorithm is able to recover the input binary data sequence, given the observation sequence.

4. IMPLEMENTATION ON THE TMS320C62

The DSP TMS320C62 was chosen because it allows a fast development using the DSP C optimized compiler. Moreover, we took advantage of specific capabilities, like implemented functions. The computational power of this DSP (8 instructions in 1 cycle time, Tc=5 ns) allows real-time implementation for reasonable filter lengths.

Implementation notes:
We give here some hints on the practical implementation of the previously presented algorithm.

Computation in the fixed-point format:
This algorithm should be implemented on a floating-point processor. But it is often less expensive to use a fixed-point processor. We then had to adapt all operations toward a fixed-point implementation.

We handle 2 types of variables: observations and probabilities. The probabilities naturally lie between 0 and 1. As far as the observations are concerned, we assume that the data are scaled between -1 and +1 during the acquisition step. We may observe that, since we have to recover a sequence of bits, such scaling may be viewed as the introduction of a scaling factor on the channel's impulse response. This is of no importance here, since knowledge of the impulse response is not needed.

In fixed-point format, the dynamic range is sufficient for our purpose. A 16-bit number can vary from -32768 to 32767, so we choose to represent variables in Q15 format. Location of the binary point affects neither the arithmetic unit, nor the multiplier in the DSP. It only affects the storage of the result, which is not a problem here, because we are able to define the dynamic range of all calculations.

On the other hand, division and exponential calculations are critical operations on a fixed point DSP. Indeed, division of two 16-bit numbers generates an integer, which implies a severe precision loss on the result. The exponential function is completely unknown by the processor since it results in a floating point number. We implemented specific methods to perform these two operations:

- Method used to perform divisions:
  Both operands are coded in 16 bit registers. Dividing a number by a bigger one would result in integer 0. To avoid this, we cast both operands in 32 bit registers and multiply the numerator by the biggest possible scaling factor in order to keep the highest precision. Then, the integer division is performed, and since we know the dynamic range of the result, we rescale the result in the chosen format in a 16-bit register.

- Method used to compute exp(x):
  We use the following remark on the exponential function:
  $$\exp(x = \sum_i a_i \cdot 2^i) = \prod_i \exp(a_i \cdot 2^i)\,,\ a_i \in \{0,1\}$$
  The 16 values of $\exp(2^i)$ are recorded in an array, which enables to reduce the calculation of exp(x) to a simple product of memory words.

Memory requirements:
We analyzed the memory requirements of this algorithm. We need to store all data $B=\{b_i(O_t)\}$, $\delta_t(i)$ and $\delta_{t-1}(i)$, $M=\{m_i\}$, $\forall i = 1..N$ and especially Ψ which is an N by T array (T being the length of the observation sequence). As in the classical Viterbi algorithm, we introduce a *truncature length* of Max(5N, 100) before backtracking to find the best path.

For a length-p transmission channel, we need 100 observation samples or less. Thus, Ψ consumes $100*N=100*2^p$ memory elements. When N is greater than 256, these memory elements are words. For an 11-coefficient channel, storage of Ψ requires 400 Kbytes.
But our target, simulating the standard DSP card, has only 64 Kbytes of internal data memory. This is the

reason why we were only able to test the algorithm on small filter lengths. However, specific board embarking the TMS320C62 can be easily designed.

Computation time:
The analysis of the algorithm shows that its complexity is O(N^2). This means that if we double the number of the channel coefficients, we increase the number of cycles needed for the processing of one observation sample by four.

Experimental results on the DSP:
All tests on the DSP were performed using the training sequence initialization of the HMM.

The algorithm was splitted in several functions, which are all linked to the main program main.c. These functions are:

Viterbi, which computes new $\delta_t(i)$ and $\Psi(t)$ for each observation.

calc_b, which calculates $B=\{b_i(O_t)\}$.

adjust, which updates the array $M=\{m_i\}$ and σ^2.

norm, which normalizes the array $\delta_t(i)$

init_m, which initializes the array $M=\{m_i\}$ using the training sequence.

expo, which computes the exponential function.

init_var, which initializes the variance σ^2 with the first means.

To assemble and compile the C program , we used the command:

cl6x -g -o -k -mg -me main.c -z lnk.cmd -l rts6201e.lib -o main.out -m main.map

We started the simulator with *sim62x -me main*.

The following table sums up the number of cycles used for a 2-coefficient channel, for the four first samples of the observation sequence.

function	number of calls	number of cycles	
		without optimization	with optim. -o2
Viterbi	4	2581	1397
calc_b	5	609	424
adjust	4	554	287
norm	5	306	140
init_m	1	218	167
expo	5	183	72
init_var	1	83	46
main		17374	10733

Optimization (compiler option -o2) enables to decrease execution time by 40% in this example.
This other table gives the number of cycles consumed for a 4-coefficient channel (first 5 observations). We also report here the effect of switch -o3 (instead -o2).

function	number of calls	number of cycles	
		optimization -o2	optimization -o3
Viterbi	5	4451	4241
calc_b	6	1474	1297
adjust	5	830	798
norm	6	643	643
init_m	1	239	239
expo	6	72	72
init_var	1	53	53
main		29980	29181

We notice that a simple processing (*Viterbi* function) lasts 4 times more with a channel twice longer.

5. CONCLUSION

We have both presented a new algorithm for the deconvolution of binary data and its implementation on a new fixed point DSP. The algorithm proved to be efficient, possibly in non-stationary and blind contexts. Concerning the DSP implementation, our goal was to examine the feasibility of a quick implementation using the new compiler and tools provided by TI. These tools proved to be very efficient, and we obtained a functional implementation in a few days. Of course, improvements and gains can still be achieved in several fields, particularly regarding reduction of redundant operations and direct assembler optimization of sensible functions.

REFERENCES

[1] L. R. Rabiner & B. H. Juang, *An Introduction to Hidden Markov Models*, IEEE ASSP Magazine, January 1986

[2] L. R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, vol. 77, no. 2, February 1989

[3] Texas Instruments, *TMSC320C6xx DSP Design Work Shop, Student guide*, April 1997

[4] Texas Instruments, *TMS320C6xx C Source Debugger*, January. 1997

[5] Texas Instruments, *Programmer's guide*, July 1997

[6] J.G. Proakis, *Digital Communications*, 3rd edition New York, Mc Graw-Hill, 1996