

# Optimum Algorithm to Minimize Human Interactions in Sequential Computer Assisted Pattern Recognition

Jose Oncina

*Dep. Lenguajes y Sistemas Informáticos,  
Universidad de Alicante, SPAIN*

---

## Abstract

Given a Pattern Recognition task, Computer Assisted Pattern Recognition can be viewed as a series of solution proposals made by a computer system, followed by corrections made by a user, until an acceptable solution is found. For this kind of systems, the appropriate measure of performance is the expected number of corrections the user has to make.

In the present work we study the special case when the solution proposals have a sequential nature. Some examples of this type of tasks are: language translation, speech transcription and handwriting text transcription. In all these cases the output (the solution proposal) is a sequence of symbols. In this framework it is assumed that the user corrects always the first error found in the proposed solution. As a consequence, the prefix of the proposed solution before the last error correction can be assumed error free in the next iteration.

Nowadays, all the techniques in the literature relies in proposing, at each step, the most probable suffix given that a prefix of the “correct” output is already known. Usually the computation of the conditional most probable output is an NP-Hard or an undecidable problem (and then we have to apply some approximations) or, in some simple cases, complex dynamic programming techniques should be used (usually some variant of the Viterbi algorithm).

In the present work we show that this strategy is not optimum when we are interested in minimizing the number of human interactions. Moreover we describe the optimum strategy that is simpler (and usually faster) to compute.

*Key words:* Computer Assisted Pattern Recognition, Minimizing Human interaction

---

---

*Email address:* [oncina@dlsi.ua.es](mailto:oncina@dlsi.ua.es) (Jose Oncina).

## 1 Introduction

Nowadays computers are increasingly applied to solve challenging Pattern Recognition tasks. However, in some cases the results are not as good as desired and human (user) supervision is mandatory. Most often this supervision is off-line: the computer produces a possible solution to the proposed task and then, the user corrects all the errors.

In the present work we are interested in Computed Assisted Pattern Recognition (CAPR) [Vidal et al., 2007]. This framework can be described as follow:

- (1) the user proposes a task to solve,
- (2) the computer proposes a solution
- (3) if the user finds the solution good enough, it stops the process
- (4) if not, the user makes a correction
- (5) the computer, taking into account the correction, proposes a new solution
- (6) return to step 3

Although the classical measure of performance is the Risk Minimization [Duda et al., 2001] and usually simplified as minimizing the error rate, almost each Pattern Recognition field has its own way to measure the performance: Bilingual Evaluation Understudy (BLEU) [Papineni et al., 2002] in translation tasks, precision and recall in information retrieval [Manning et al., 2008], Word Error Rate (WER) in speech recognition [Jelinek, 1997] and so on. In CAPR, as the most valuable resource is the time a user has to spend in the task, an adequate measure of performance is the expected number of corrections the user makes [Vidal et al., 2007].

In this work we restrict ourselves to the case when the solution has a sequential nature (can be expressed as a string). Moreover, we assume the user always corrects the first error he finds. Then, the solution prefix before the last correction can be assumed error free for the next iteration.

Let us see three examples of Sequential Computer Assisted Pattern Recognition tasks:

- Human language translation [Civera et al., 2006, Tomás and Casacuberta, 2006, Civera et al., 2004, Foster, 2002, Foster et al., 1997, Vidal et al., 2006]: in this case, given an input sentence, we can assume the user reads the proposed translation and always corrects the first word that does not fit.
- Speech to text transcription [Rodríguez et al., 2007]: given a speech, the user always corrects the first word that does not match with the listening.
- Handwriting text transcription [Romero et al., 2007, Toselli et al., 2007]: given the image of the text, the user corrects the first wrong word.

Translate:	En un lugar de la Mancha, de cuyo nombre no quiero acordarme, ...
Computer:	<i>In a place of La Mancha, of whose name I do not want to decide to me, ...</i>
User:	In a <b>village</b>
Computer:	<i>In a village of La Mancha, of whose name I do not want to decide to me, ...</i>
User:	In a village of La Mancha, <b>the</b>
Computer:	<i>In a village of La Mancha, the name of which I do not want to decide to me, ...</i>
User:	In a village of la Mancha, the name of which I <b>have</b>
Computer:	<i>In a village of La Mancha, the name of which I have no desire to call to mind, ...</i>
User:	In a village of La Mancha, the name of which I have no desire to call to mind, ... [OK]

Fig. 1. Example of an hypothetical Computer Assisted Translation, *italic* text represents a computer guess, **bold** text represents user corrections.

Figure 1 shows an hypothetical application example of this scheme in a Spanish to English translation task. It should be noted that the correction of an error may lead to automatic correction of subsequent errors. Then, although in the first attempt there were eleven translation errors, the text was satisfactorily translated making only three corrections.

There exists some systems that work using this scheme. Most of them (conceptually) work as follows: given the task (*i.e.* the whole sentence to translate) and the already known error free solution prefix, on each iteration, first they build a probabilistic representation (typically a word-graph) of all the possible solutions that begin with the error free prefix. In a following step, they propose the most probable solution.

Although this seems a very sensible strategy, in this work we show that it is not optimum if the objective is to minimize the number of human interactions. In fact, we are going to describe the optimum one.

Moreover, depending on the probabilistic representation (*i.e.* smoothed n-grams), in order to find the most probable solution one has usually to face NP-Hard or undecidable problems (and then, use approximations) or to resort to complex Dynamic Programming techniques in the most simple cases (usually some variants of the Viterbi algorithm). Our algorithm is a greedy technique and, if any of the usual probabilistic representations is used, is free of those inconveniences.

## 2 The algorithm

As stated in the introduction, the problem can be reduced to find a strategy that, given an input (*i.e.* a sentence to translate), given a probabilistic model (*i.e.* a word-graph describing the probabilities of all the possible translations of the sentence), and given the evidence (the error free prefix), provides a solution

that will minimize the expected number of interactions when the strategy is applied repeatedly.

Given that the solutions in our framework have a sequential structure, we are going to represent them by strings where each symbol represents a word (*i.e.* in a translation task), a character (*i.e.* in a handwriting transcription task), a note (*i.e.* in a score transcription task), or some more complex structure depending on the task.

Let  $\Sigma$  be a set of symbols (*vocabulary*)<sup>1</sup>, let  $\Sigma^*$  be the set of all the strings that can be formed concatenating symbols of  $\Sigma$  and let us denote the empty string by  $\lambda$ . In the following, the letters  $a, b, \dots \in \Sigma$  will denote individual symbols and letters  $x, y, \dots$  strings in  $\Sigma^*$ . As usual,  $xa$  ( $xy$ ) will denote the concatenation of string  $x$  with the symbol  $a$  (string  $y$ ). The probabilistic model will be represented by a probability distribution over the strings ( $p : \Sigma^* \rightarrow [0, 1]$ ), that is, a function that assigns to each string a probability. Of course,

$$\sum_{x \in \Sigma^*} p(x) = 1$$

Then, the problem can be stated as follows: suppose a hidden string has been drawn from the probability distribution  $p$  (*i.e.* given a sentence to translate, the “correct” translation that the user has into his mind and the system has to guess), we have to devise an algorithm that could guess which string is. To obtain information, the algorithm can resort to a function (*oracle*)  $O : \Sigma^* \rightarrow \Sigma^* \cup \{\mathbf{YES}\}$ , such that, given a string  $x$ ,  $O(x)$  returns:

- **YES**: if  $x$  is the hidden string.
- Otherwise, the longest prefix of  $x$  that is also prefix of the hidden string, concatenated with the next symbol of the hidden string. (That is, the error free prefix of  $x$  concatenated with the correction made by the user.)

Note that each use of the  $O(\cdot)$  function represents an interaction with the user.

The objective is to find the hidden string making as fewer uses of the  $O(\cdot)$  function as possible.

---

<sup>1</sup> In order to simplify the notation, to mark the end of strings we are going to assume that, in every vocabulary, it exists an special symbol ( $\#$ ) that only appears at the end of every string.

```

 $x = \lambda$  //  $x$  is the error free prefix
repeat
   $y = S(x)$  //  $y$  is the proposed solution using evidence  $x$ 
   $x = O(y)$  //  $x$  is the new error free prefix
until  $x == \mathbf{YES}$ 

```

Fig. 2. Interaction scheme

## 2.1 The Scheme

Let us define a *strategy* as a function  $S : \Sigma^* \rightarrow \Sigma^*$  such that, given a string (the error free prefix or evidence) it returns another string that represents a possible solution. The guessing process can be described by the schema in figure 2.

The problem now is to compute the expected number of interactions and to find the strategy that minimizes it.

In order to do so, an equivalent problem is defined. The idea is to move forward symbol by symbol and count, as interaction, only when a mistake is committed. Then the strategy for this new problem is defined as a function  $S_s : \Sigma^* \rightarrow \Sigma$  such that given the evidence (the error free prefix of the last iteration) it returns a guess for the next symbol.

Note that an strategy for the former problem can be easily defined in base to the new strategy function as:

$$S(x) = xa_1 \dots a_n \quad \text{where } a_i = S_s(xa_1 \dots a_{i-1}) \quad \text{and } a_n = '\#' \quad (1)$$

To adapt the oracle to this scheme it is redefined as a function  $O_s : \Sigma^* \rightarrow \Sigma \cup \{\mathbf{YES}\} \cup \{\mathbf{PREF}\}$ . Given a string  $x$ ,  $O_s(x)$  returns:

- (1) **YES**: if  $x$  is equal to the hidden string
- (2) **PREF**: if  $x$  is a prefix of the hidden string
- (3) a symbol  $a$ : if  $x$  is not a prefix of the hidden string, but when the last symbol of  $x$  is replaced by  $a$  it becomes a prefix of the hidden string (this case models the correction of an error.)

Figure 3 depicts the symbol to symbol schema.

Note that, in this schema, since  $x$  is the previous error free prefix concatenated with the proposed symbol, the only place where an error can appear is in the last symbol and then, no other possibilities should be taken into account in the definition of  $O_s(\cdot)$ .

```

x = λ // x is the error free prefix
loop
  a = S_s(x) // a is the proposed new symbol using evidence x
  b = O_s(xa)
  if(b == YES) exit loop // we have the correct answer
  if(b == PREF) x = xa // we have guessed the next symbol
  else x = xb // we made a mistake; xb is the next error free prefix
end loop

```

Fig. 3. Symbol to symbol interaction schema

Now the objective is to minimize the expected number of times that the  $O_s(\cdot)$  function returns a symbol (corrects an error). Obviously, the number of interactions in this case and in the previous one are exactly the same.

Suppose now that, when considering prefix  $x$ , the strategy says that  $a$  should be the next symbol (that is  $S_s(x) = a$ ). Let us denote by  $A_S(x)$  the expected number of error corrections when using strategy  $S$  and evidence  $x$ , and let us denote by  $p(a|x)$  the probability that the symbol that follow the (error free) prefix  $x$  is  $a$ . When computing  $A_S(x)$ , if  $a$  is the next correct symbol (probability  $p(a|x)$ ), then all the error correcting operations will be performed while guessing the symbols of the remaining hidden string ( $A_S(xa)$ ). If the correct next symbol is  $b \neq a$  (probability  $p(b|x)$ ) we have to count an error plus the errors committed guessing the rest of the symbols of the hidden string ( $1 + A_S(xb)$ ) (Note that the symbol  $\#$  is used as an end-of-string mark; evidently,  $A_S(x\#) = 0$  for any strategy  $S$ .) Then, the expected number of error corrections can be computed as:

$$A_S(x) = p(a|x)A_S(xa) + \sum_{b \neq a} p(b|x) (1 + A_S(xb)) \quad (2)$$

$$= \sum_{b \neq a} p(b|x) + \sum_b p(b|x)A_S(xb) \quad (3)$$

$$= (1 - p(a|x)) + \sum_b p(b|x)A_S(xb) \quad (4)$$

It can be seen that the second term of equation 4 does not depends of which symbol the strategy selects when considering evidence  $x$ , and the first term is minimized if the strategy consists in choosing the symbol  $a$  that maximizes the probability  $p(a|x)$ .

Then, returning to the original problem and using equation 1, given an evidence  $x$  the optimum choice is the strategy

$$S(x) = xa_1 \dots a_n \quad \text{where } a_i = \operatorname{argmax}_{a \in \Sigma} p(a|xa_1 \dots a_{i-1})$$

instead of the classical one

$$S(x) = xy \quad \text{where } y = \operatorname{argmax}_{y \in \Sigma^*} p(y|x)$$

## 2.2 Complexity issues

The complexity of both algorithms depends on the representation of the probabilistic model. The complexity of the classical algorithm is exactly the complexity of computing  $y^* = \operatorname{argmax}_{y \in \Sigma^*} p(y|x)$  and the complexity of the optimum algorithm is  $n$  times the complexity of computing  $p(a|x)$ , where usually  $n \sim |y^*|$  and, in the practice,  $n$  is bounded by the maximum number of symbols that can be presented to the user at the same time.

Let us see some examples:

- If the probabilistic model is implemented as a finite set of words (like in the experiment in section 4.1), both probabilities can be precomputed and then, both algorithms have the same complexity ( $O(1)$ ).
- If the probabilistic model is implemented as a *Stochastic Deterministic Finite Automaton* or similar (*i.e.* n-grams) the probability of the next symbol is just the probability that labels each edge and then it can be computed in constant time, but to find the most probable suffix one has to recur to some variant of the Viterbi algorithm ( $O(|y^*||E|)$  where  $|E|$  is the number of edges in the automaton).
- If the probabilistic model is implemented as a *Nondeterministic Stochastic Finite Automaton* or similar (*i.e.* some smoothed n-grams models), the most probable next symbol can be computed in  $O(|E|)$  (where  $|E|$  is the number of edges in the automaton) but finding the most probable suffix becomes an NP-Hard.

## 3 Examples

### 3.1 Three strings example

Suppose, as shown in figure 4, we have only three strings with non-zero probability.

Let us compute the expected number of iterations in both cases, using the classical strategy and using the optimum one.

string	probability
<i>aa</i>	0.36
<i>ab</i>	0.24
<i>b</i>	0.4

Fig. 4. Probability distribution

In the classical strategy the algorithm, at the beginning, when no hidden string prefix is known, proposes the most probable string, that is, the string *b* ( $p(b) = 0.4$ ). In using this proposal it guesses correctly the hidden string for the 40% of the cases, with no corrections. When the hidden string is not *b* the user makes one correction and returns *a* as prefix (the two other possible strings begins by *a*). In this case, the most probable string that begins by *a* is *aa* ( $p(aa) = 0.36$ ) then the algorithm tries *aa* and guesses correctly the 36% of times making one mistake. In the remaining 24% of the cases, when the hidden string is *ab*, it makes two mistakes.

Then, the expected number of mistakes is:

$$\begin{aligned}
 A &= 0 \cdot p(b) + 1 \cdot p(aa) + 2 \cdot p(ab) \\
 &= 0 \cdot 0.4 + 1 \cdot 0.36 + 2 \cdot 0.24 \\
 &= 0.84
 \end{aligned}$$

If the optimum strategy is used, we have first to compute:

$$\begin{aligned}
 p(a|\lambda) &= \frac{p(aa) + p(ab)}{p(aa) + p(ab) + p(b)} = 0.6 \\
 p(b|\lambda) &= \frac{p(b)}{p(aa) + p(ab) + p(b)} = 0.4 \\
 p(a|a) &= \frac{p(aa)}{p(aa) + p(ab)} = 0.6 \\
 p(b|a) &= \frac{p(ab)}{p(aa) + p(ab)} = 0.4
 \end{aligned}$$

Following this strategy, the first string to try is *aa* ( $p(a|\lambda) > p(b|\lambda)$  and  $p(a|a) > p(b|a)$ ). Then it guesses correctly the hidden string 36% of the times with no mistakes. Now, if the hidden string is not *aa*, there are two possibilities:

- if the hidden string is *ab* (24% of the times) the user makes a correction and returns *ab*.
- if the hidden string is *b* (40% of the times) the user makes one correction and returns *b*.

Stochastic Regular Grammar	$p(\lambda) = 0.1$	$p(\lambda) = 0.1$
0.9 $S \rightarrow aS$	$p(a) = 0.09$	$p(a \lambda) = 0.9$
0.1 $S \rightarrow \lambda$	$p(aa) = 0.081$	$p(a a) = 0.9$
	...	...

Fig. 5. Stochastic Regular Grammar and some string probabilities

In both cases the process ends at next iteration.

The expected number of error corrections can be computed as:

$$\begin{aligned}
A &= 0 \cdot p(aa) + 1 \cdot p(ab) + 1 \cdot p(b) \\
&= 0 \cdot 0.36 + 1 \cdot 0.24 + 1 \cdot 0.4 \\
&= 0.64
\end{aligned}$$

As can be checked, the expected number of error corrections is lower than using the classical strategy.

### 3.2 Infinite string length example

When building a string by concatenating the most probable next symbol, depending on the probabilistic model, it is possible to fall in an infinite loop. Let us see an example.

Suppose the probabilistic model is implemented as the Stochastic Regular Grammar depicted in fig 5.

It is easy to see that, using the classical strategy, the algorithm will guess correctly the string  $\lambda$  ( $p(\lambda) = 0.1$ ) with no corrections, the string  $a$  ( $p(a) = 0.09$ ) with one correction, the string  $aa$  ( $p(aa) = 0.081$ ) with two corrections, and so on.

Then the expected number of corrections can be computed as:

$$A = \sum_{i=0}^{\infty} p(a^i) i = 9$$

Using the optimum algorithm, the algorithm first proposes an infinite length string of  $a$ 's<sup>2</sup>. As such string has a zero probability the algorithm will fail to guess the hidden string in its first attempt, but the only way to make a

<sup>2</sup> Note that this is not a problem from a practical point of view, it is enough to return a string sufficiently long

correction is by providing the end-of-string point. Then, the algorithm guesses correctly the hidden string with just one error.

## 4 Experiments

In order to check experimentally the correctness of our approach, two sets of experiments were designed.

### 4.1 *Pool of words experiment*

In the first set of experiments a finite distribution of probability was built as follows: first a database of 100,000 strings of length 20 were randomly generated by concatenating symbols extracted with equal probability from a finite vocabulary. Then a probability was assigned randomly (uniform distribution) to each string.

In the experiment, strings were extracted using this distribution and three different strategies were tested: the classical, the optimum and a base line strategy that consisted on proposing the first suffix, found in the databases that is consistent with the evidence.

For each database the number of mistakes were counted and averaged for 1000 extractions. The experiment was repeated for 10 different databases and the standard deviations were computed. The experiments were repeated for increasing size vocabularies from 2 to 1024. The results are summarized in figure 6.

As expected, it can be observed that in all cases our strategy commits, in average, less mistakes than other strategies. It can also be observed that, in this setting, the expected values of the three strategies decreases as the vocabulary size increases. The reason for this is that the chance to have overlapping strings decreases very quickly as the vocabulary increases, provoking that a strings can be identified by just seeing a few initial symbols.

### 4.2 *Feldman task*

The algorithm has been also tested with an extension [Castellanos et al., 1998, 1994] of a pseudo-natural task proposed by Feldman et al. [Feldman et al., 1990]. The original task consisted of descriptions of simple two-dimensional visual scenes involving a few geometric objects with different shape, shade and

Voc	Base line		Classical		Optimum		Classical / Greedy
	avrg	dev	avrg	dev	avrg	dev	
2	8.49	0.09	7.90	0.05	7.53	0.07	95.23
4	6.48	0.03	6.09	0.02	5.86	0.05	96.25
8	5.17	0.04	4.89	0.02	4.75	0.02	97.00
16	4.25	0.02	4.04	0.03	3.96	0.03	97.89
32	3.63	0.01	3.45	0.02	3.40	0.02	98.62
64	3.09	0.02	2.99	0.01	2.95	0.01	98.59
128	2.83	0.01	2.72	0.01	2.71	0.01	99.54
256	2.48	0.01	2.35	0.01	2.34	0.01	99.76
512	2.16	0.01	2.10	0.01	2.09	0.01	99.48
1024	2.04	0.00	2.01	0.01	2.00	0.01	99.40

Fig. 6. Pool of words experiment

a large square and a large circle are far to the right of a medium dark triangle and a circle
the dark circle which is bellow the circle and the triangle is removed
a small dark circle is added far above the medium dark circle and the medium circle

Fig. 7. Some sentences from the Extended Feldman task.

size, and located in different relative positions. The original language of this task was extended to cover the possibility of adding or removing objects to or from a scene (see fig. 7 for some examples).

The Context Free Grammar that describes the language was transformed in a probabilistic one by distributing the probability mass uniformly over all the rules sharing the same left-hand non terminal.

From the probability distribution induced by the Probabilistic Context Free Grammar 1000 random sentences were extracted and both strategies, the classical and the optimum one, were used to guess the strings.

The average number of corrections for both strategies is shown in figure 8.

Once more the experiments show the superiority of the optimal strategy.

Strategy	Corrections
Classical	8.03
Optimum	6.39

Fig. 8. Feldman task experiment, average number of corrections over 1000 sentences

## 5 Conclusions and future work

We have proposed an optimum strategy to minimize the expected number of interactions in Sequential Computer Assisted Pattern Recognition tasks. In this work, two different sets of experiments were also performed to illustrate the validity of the theoretical work.

The proposed algorithm, as well as being optimum, can be implemented as a simple greedy algorithm. The classical strategy usually involves complex dynamic programming techniques or should recur to approximations when the probability model is complex.

In the future we plan to apply this strategy on more complex tasks.

## Acknowledgements

This work was partially supported by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018). This work was partially supported by the Spanish Ministerio de Educación y Ciencia through grant DPI2006-15542-C04-01. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

## References

- A. Castellanos, I. Galiano, and E. Vidal. Applications of OSTIA to machine translation tasks. In *Grammatical Inference and Applications*, LNAI, pages 93–105, Campello, Alicante, Spain, september 1994. 2nd International Colloquium on Grammatical Inference, Springer-Verlag.
- A. Castellanos, E. Vidal, A. Varó, and J. Oncina. Language understanding and subsequential transducer learning. *Computer Speech and Language*, 12: 193–228, 1998.
- J. Civera, J.M. Vilar, E. Cubel, A.L. Lagarda, S. Barrachina, F. Casacuberta, E. Vidal, D. Picó, and J. González. A syntactic pattern recognition approach

- to computer assisted translation. In *Advances in Statistical, Structural and Syntactical Pattern Recognition*, LNCS. Springer, 2004.
- J. Civera, J.M. Vilar, E. Cubel, A.L. Lagarda, S. Barrachina, F. Casacuberta, and E. Vidal. A novel approach to computer assisted translation based on finite-state transducers. language processing. In *Finite-State Methods and Natural Language Processing. Revised papers of Proceedings of FSMNLP 2005*, volume 4002 of *LNAI*, pages 32–42. Springer, 2006.
- R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2001.
- J. A. Feldman, G. Lakoff, A. Stolke, and S. H. Weber. Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-009, International Computer Science Institute, Berkeley, CA, USA, 1990.
- G. Foster. *Text Prediction for Translators*. PhD thesis, Université de Montréal, May 2002.
- G. Foster, P. Isabelle, and P. Plamondon. Target-text mediated interactive machine translation. *Machine Translation*, 12(1–2):175–194, 1997.
- F. Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, 1997.
- C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press. 2008.
- K. Papineni, S. Roukos, T. Ward and W.J. Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. In: *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA (2002) 311–318
- L. Rodríguez, F. Casacuberta, and E. Vidal. Computer assisted transcription of speech. volume 4477 of *LNCS*, pages 241–248, Girona (Spain), June 2007. Springer.
- V. Romero, A. H. Toselli, L. Rodríguez, and E. Vidal. Computer assisted transcription for ancient text images. In *In International Conference on Image Analysis and Recognition (ICIAR 2007)*, volume 4633 of *LNCS*, pages 1182–1193, Montreal (Canada), August 2007. Springer.
- J. Tomás and F. Casacuberta. Statistical phrase-based models for interactive computer-assisted translation. In *Proceedings of the Coling/ACL2006*, pages 835–841, Sydney, Australia, July 2006.
- A. H. Toselli, V. Romero, L. Rodríguez, and E. Vidal. Computer assisted transcription of handwritten text. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 944–948, Curitiba, Paraná (Brazil), September 2007. IEEE Computer Society.
- E. Vidal, F. Casacuberta, L. Rodríguez, J. Civera, and C. Martínez. Computer-assisted translation using speech recognition. *IEEE Transaction on Audio, Speech and Language Processing*, 14(3):941–951, 2006.
- E. Vidal, L. Rodríguez, F. Casacuberta, and I. García-Varea. Computer assisted pattern recognition. In *Proceedings of the 4th Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms*, page to be published, Brno, Czech Republic, June 2007.