université
PARIS-SACLAY

M2 Data Science

Probabilistic Graphical Models

Mini Project
Barcode decoding with HMMs

David Obst
Mariam Barry
Firas Jarboui
Imad El Hanafi
Promotion 2018

Professeur :
M. Umut Şimşekli

29 November 2017

Contents

1 Introduction

For most companies which receive and ship, being able to efficiently decode barcodes is of paramount importance. While laser scanners remain widely in use, they are tedious to use in practice and expensive to implement. That is why the use of cameras to do this task has soared the past years. Indeed, they are much more efficient, less cumbersome to use and can decode barcodes oriented in any direction. Despite presenting those advantages, cameras also have their share of drawbacks. In practice due to noise and blur the image of the observed scanline can be degrated, making the task difficult.

That is why the aim of this project is to develop a probabilistic model for modeling a specific type linear barcodes, called UPC-A. Our goal is to infer the subsequent digits given a barcode scanline that is obtained from a gray-scale image. In order to achieve that, we will incorporate the sequential and hierarchical information of the barcodes into a single dynamic bayesian network, namely a hierarchical Hidden Markov Model (H-HMM). We will model different layers of the hierarchy of barcodes by using an HMM, in particular UPC-A barcodes encoding.

2 Description of the model

The first point in such a task is always to define the model. The latter is explicited by the subject and involves 4 variables:

- The only observed variable is the intensity of the observed pixel x_n in the scaline ($n \in \{1, \dots, N\}$ corresponds to the pixel number). As usual in image tasks, this intensity is between 0 (white) and 255 (black). While in a perfect scaline picture, the value would be binary (either perfectly black or white), in our case it is degraded and is what makes the task challenging. The probability distribution of x_n is governed by the following latent variables.

- The symbol s_n to which pixel x_n belongs to. Symbols can either be the digits (between 0 and 9) or the special symbols which are the starting and ending quiet zones, the start or end of the code and the middle guard.

- A counter variable c_n. It gives the position of x_n in the symbol s_n. Therefore it is an integer between 1 and the length $\ell(s_n)$ of the symbol.

- The index m_n of the symbol. In our model $m_n \in \{1, \dots, 6\}$ since there will be 6 digits left and right of the middle guard.

2.1 Directed Acyclic Graph

Question 1 - Therefore following the different probability distributions given by the subject, we can obtain the directed acyclic graph for our model (given fig. 2.1).

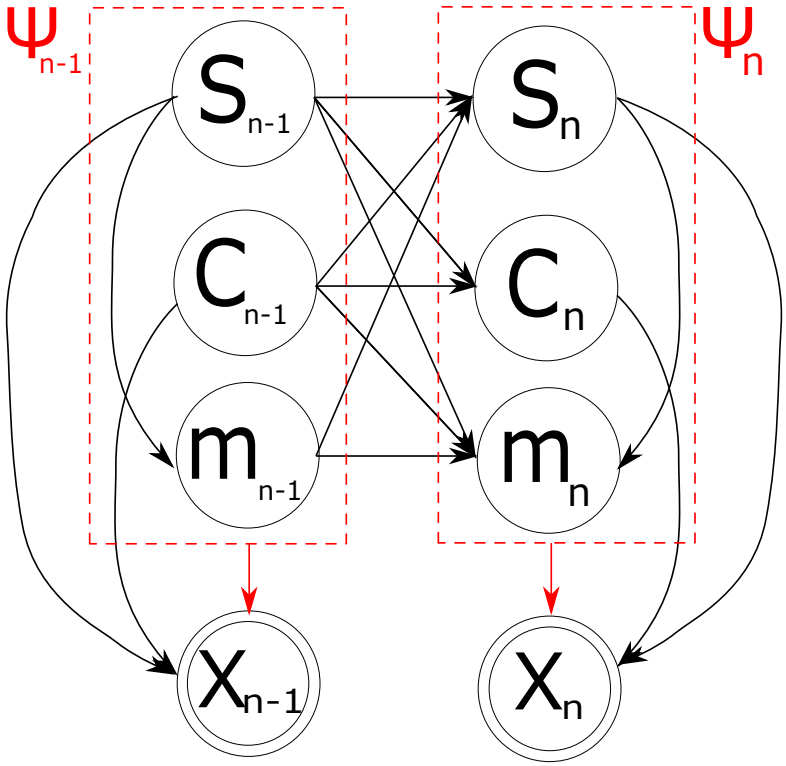


Figure 1: Graphical Model

2.2 Definition of the HMM

Question 2 - Up to now, we do not have a clearly identified hidden Markov chain (HMM). Therefore we put the latent variables s_n, c_n an m_n together in a hidden variable Ψ_n which becomes the unique hidden variable of our HMM, while x_n is still the observed one of course.

The set of all possible states of the triplet $\Psi_n = [s_n, m_n, c_n]$ can then be listed in a vector and the state of the system at the pixel n can be represented as $\Psi_n = \Omega(j)$ where j is in $1, 2, ..., (SxMxC)$ where S, M and C are respectively the number of symbols, the number of indexes and the biggest possible size for a symbol.

The transition matrix

$$A(i,j) = p(\Psi_{n+1} = \Omega(i) | \Psi_n = \Omega(j)) \tag{2.1}$$

is then coded on Matlab using the different probability distributions given in the subject (more precisely equations (1) to (5)). While be will not detail everything in this report, we give a few examples:

- If $s_{n-1} = 1$ (we are in the staring quiet zone), with uniform probability the next symbol s_n is either still the starting quiet zone (1) or the start code (3).
- If s_{n-1} is the start code, then next symbol is necessarly a (left) digit, *i.e.* $s_n \in \{6, \ldots, 15\}$. Again the probability is uniform since it can be any of the digits.

- There are also deterministic transitions: if $s_{n-1} = 2$, which means we are at the ending quiet zone, then s_n will always remain equal to 2.

Those transitions are implemented in the code. In order to check our work, we verify that the sum of all columns is indeed 1 (stochastic matrix), which is the case.

The resulting observation model is described by the variable x_n

Question 3 - The observed pixel intensity x_n has a probability distribution which is conditional to the symbol s_n and the current position within the symbol c_n. Since we are considering noisy scanline, a natural probability distribution is the gaussian one. However the distribution of the intensity should depend on either if a white pixel is expected or a black one. This is why we use a product of gaussian with two different means and the same standard deviation in function of which color is awaited (cf. equation 2.2).

$$p(x_n|s_n, c_n) = \prod_{i=0}^{1} \mathcal{N}(x_n|\mu_i, \sigma_i^2)^{1\left(f(s_n,c_n)=i\right)} \tag{2.2}$$

Concerning the choice of the means, we take $\mu_0 = 20$ and $\mu_1 = 250$ which respectively correspond to white and black. As stated above, we take identicals standard deviations for both gaussians $\sigma_0^2 = \sigma_1^2 = 5$. Fig. 2 represents a simulated x_n.

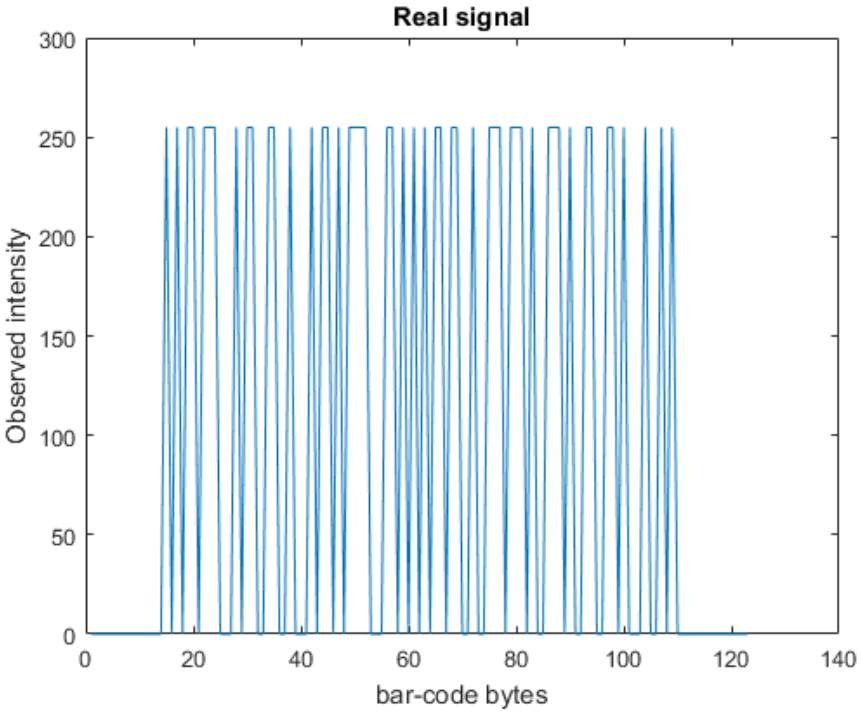


Figure 2: Simulated scanline

3 HMM inference

Now that the transition matrix is implemented, we can perform HMM inference to reconstruct the code from the observations. We consider two methods: the forward-backward algorithm seen in class as well as the Viterbi algorithm.

3.1 Forward-Backward algorithm:

The forward-backward algorithm consists of 3 major steps:

1. The forward pass, calculating the filtering distribution $p(\Psi_n, x_{1:n}) \propto p(\Psi_n|x_{1:n})$.
2. The backward pass, calculating the probability $p(x_{n+1:N}|\Psi_n)$ of the remaining observations for a given state Ψ_n.
3. Calculation of the smoothed values $p(\Psi_n, x_{1:N})$.

3.1.1 Filtering Distribution

Question 5 - The filtering distribution defined as $p(\Psi_n|x_{1:n})$ can be computed using the forward algorithm. We define the following quantities

$$\alpha_{k|k-1}(\Psi_k) = p(x_{1:k-1}, \Psi_k)$$

$$\alpha_{k|k}(\Psi_k) = p(x_{1:k}, \Psi_k)$$

We marginalize over Ψ_{k-1} we get

$$\begin{aligned} \alpha_{k|k-1}(\Psi_k) &= \textstyle\sum_{\Psi_{k-1}} p(x_{1:k-1}, \Psi_k, \Psi_{k-1}) \\ &= \textstyle\sum_{\Psi_{k-1}} p(\Psi_k|\Psi_{k-1}, x_{1:k-1}) p(\Psi_{k-1}, x_{1:k-1}) \end{aligned}$$

By d-separation we have $p(\Psi_k|\Psi_{k-1}, x_{1:k-1}) = p(\Psi_k|\Psi_{k-1})$.
Furthermore $p(\Psi_{k-1}, x_{1:k-1}) = \alpha_{k-1|k-1}(\Psi_{k-1})$
Which yields

$$\boxed{\alpha_{k|k-1}(\Psi_k) = \sum_{\Psi_{k-1}} p(\Psi_k|\Psi_{k-1}) \alpha_{k-1/k-1}(\Psi_{k-1})} \tag{3.1}$$

On the other hand, we have by d-separation:

$$\begin{aligned} \alpha_{k|k}(\Psi_k) &= p(x_{1:k}, \Psi_k) \\ &= p(x_k|\Psi_k, x_{1:k-1}) p(\Psi_k, x_{1:k-1}) \\ &= p(x_k/\Psi_k) \alpha_{k|k-1}(\Psi_k) \end{aligned}$$

Thus

$$\boxed{\alpha_{k|k}(\Psi_k) = p(x_k|\Psi_k) \alpha_{k|k-1}(\Psi_k)} \tag{3.2}$$

We can now get an expression of the filtering distribution using the Bayes formula.

$$\boxed{p(\Psi_n|x_{1:n}) = \frac{p(\Psi_n, x_{1:n})}{p(x_{1:n})} = \frac{\alpha_{n|n}(\Psi_n)}{\sum_{\Psi_n} \alpha_{n|n}(\Psi_n)}} \tag{3.3}$$

Where we used $p(x_{1:n}) = \sum_{\Psi_n} p(\Psi_n, x_{1:n}) = \sum_{\Psi_n} \alpha_{n/n}(\Psi_n)$

In practice: Knowing the transition and the transmission probabilities, we get $\alpha_{n/n}$ and $\alpha_{n/n-1}$ step by step (forward) starting from $\alpha_{1/0}(\Psi_1) = p(\Psi_1)$. After these computations, the filtering probability is given using previous formula (3). This allows us to complete (part 4) of the code.

Since $\Psi_n = (s_n, m_n, c_n)$, the distributions $p(s_n|x_{1:n})$,$p(m_n|x_{1:n})$ and $p(c_n|x_{1:n})$ are computed from $p(\Psi_n|x_{1:n})$ by marginalizing over the other two left variables. A plot is given fig. 16 of those for a case with a reasonable amount of noise ($\sigma^2 = 60$).

3.1.2 The smoothing distribution

Question 6 : The smoothing distribution defined as $p(\Psi_n|x_{1:N})$ can be computed using the forward-backward recursions. As we've seen in the class, we define the following quantities

$$\beta_{k|k+1}(\Psi_k) = p(x_{k+1:N}|\Psi_k)$$

$$\beta_{k|k}(\Psi_k) = p(x_{k:N}|\Psi_k)$$

We get the following backward recursions using the d-separation

$$\boxed{\beta_{k|k+1}(\Psi_k) = \sum_{\Psi_{k+1}} \beta_{k+1/k+1}(\Psi_{k+1}) p(\Psi_{k+1}|\Psi_k)} \tag{3.4}$$

$$\boxed{\beta_{k/k}(\Psi_k) = p(x_k/\Psi_k)\beta_{k/k+1}(\Psi_k)} \tag{3.5}$$

Then the expression of the smoothing distribution is obtained using the Bayes formula and the d-separation as follows:

$$\begin{aligned} p(\Psi_n|x_{1:N}) &= \tfrac{p(\Psi_n, x_{1:N})}{p(x_{1:N})} \propto p(\Psi_n, x_{1:N}) \\ &= p(x_{n+1:N}|\Psi_n x_{1:n}) p(x_{1:n}, \Psi_n) \\ &= p(x_{n+1:N}|\Psi_n) p(x_{1:n}, \Psi_n) \\ &= \beta_{n/n+1}(\Psi_n)\alpha_{n|n}(\Psi_n) \end{aligned}$$

In practice: Knowing the transition and transmission probabilities, we get $\beta_{n|n+1}$ and $\beta_{n|n}$ step by step (backward recursion) starting from $\beta_{N|N+1}(\Psi_N) = 1$.

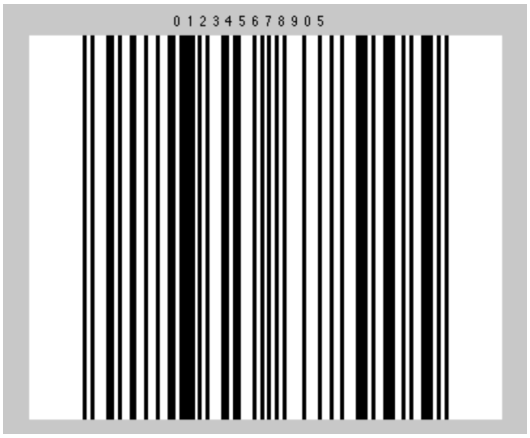
After getting α and β (forward-backward), the smoothing distributions can be computed using the previous formula (and fill part 5 of the code).

3.1.3 Test

Now we have the smoothing distribution, we can test our algorithm. We generate an observation $x_{1:N}$ (a barcode).

We run the forward-backward algorithm and we get the smoothing distribution

$$p(\Psi_n|x_{1:N})$$



A barcode (observation)

. Then for every n, the hidden state will be the one maximizing the smoothing probability (the most probable). This allows us to construct the hidden observations $\Psi_n = (s_n, m_n, c_n), n \in \{1, N\}$ and then plot them.

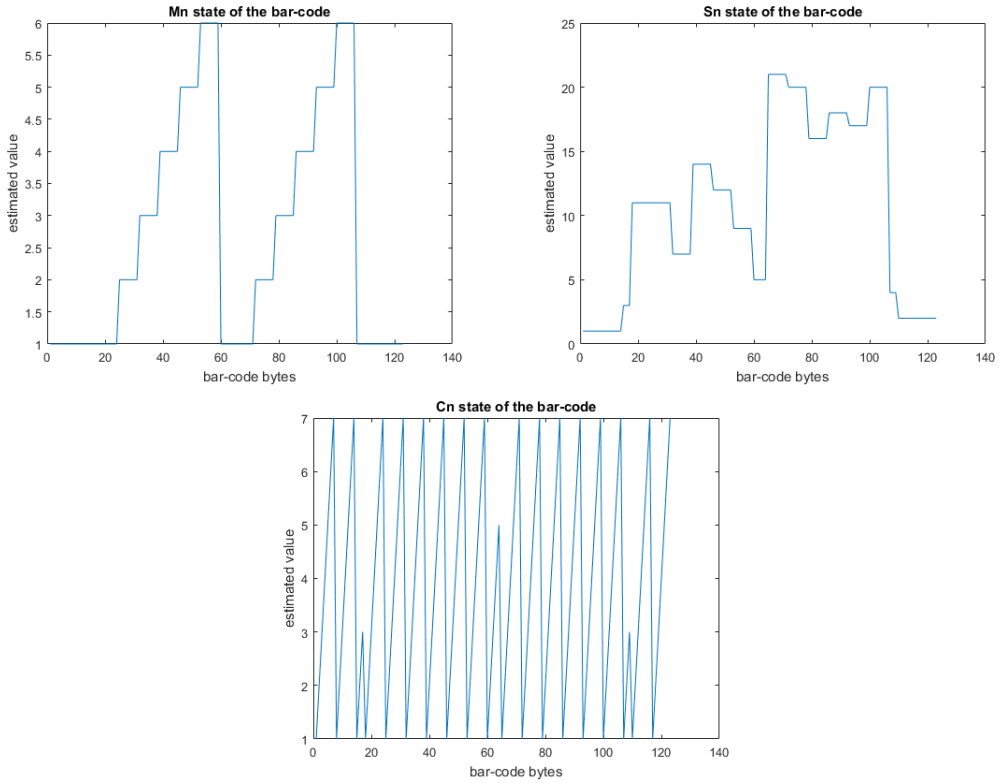


Figure 4: Plot of the three latent variables in function of pixel number n

As a conclusion, our forward-backward algorithm allows us to predict the hidden state based on observations x_n.

3.2 Viterbi algorithm:

- Question 7 : The goal of the Viterbi algorithm is to compute the most likely path

$$\Psi^*_{1:N} = \underset{\Psi_{1:N}}{\operatorname{argmax}}\, p(\Psi_{1:N}|x_{1:N}) = \underset{\Psi_{1:N}}{\operatorname{argmax}}\, p(\Psi_{1:N}, x_{1:N})$$

For a given $n \in \{1, 2, \ldots, N\}$, one has

$$\begin{aligned}\max_{\Psi_{1:n}} p(\Psi_{1:n}, x_{1:n}) &= \max_{\Psi_{1:n}} p(x_n|\Psi_n)p(\Psi_n|\Psi_{n-1})p(\Psi_{1:n-1}, x_{1:n-1}) \quad \text{(can be shown by using d-sep}\\ &= \max_{\Psi_n} \max_{\Psi_{1:n-1}} p(x_n|\Psi_n)p(\Psi_n|\Psi_{n-1})p(\Psi_{1:n-1}, x_{1:n-1})\end{aligned}$$

Let's define

$$\mu_n(\Psi_n) = \max_{\Psi_{1:n-1}} p(\Psi_{1:n}, x_{1:n})$$

One can see that one has

$$\mu_n(\Psi_n) = \max_{\Psi_{n-1}} p(x_n|\Psi_n)p(\Psi_n|\Psi_{n-1}) \max_{\Psi_{1:n-2}} p(\Psi_{1:n-1}, x_{1:n-1})$$

And therefore

$$\boxed{\begin{cases}\mu_n(\Psi_n) = \max_{\Psi_{n-1}} p(x_n|\Psi_n)p(\Psi_n|\Psi_{n-1})\mu_{n-1}(\Psi_{n-1}) \quad \text{for } n \geq 2\\ \\ \mu_1(\Psi_1) = p(x_1|\Psi_1)p(\Psi_1)\end{cases}}$$

These equations mean that for each step n and for each possible state Ψ_n, we find the maximum path which ends at Ψ_n (just by linking one of the previous paths to Ψ_n). Fig. 5 illustrates the principle of those steps.

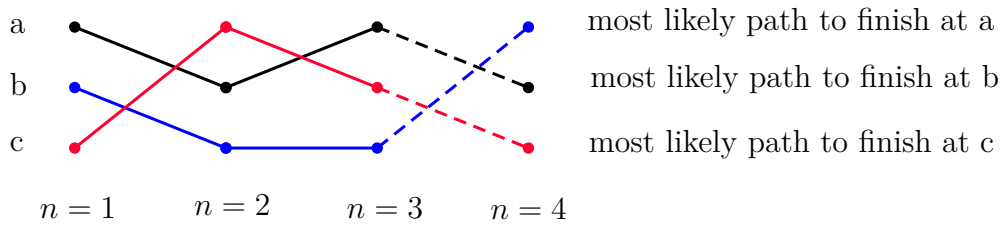


Figure 5: Principle of the Viterbi algorithm illustrated on a lattice graph

At every step n and for every possible state Ψ_n, one keeps track of the maximizing state Ψ_{n-1} which gives $\mu_n(\Psi_n)$:

$$\Gamma(\Psi_n, n) = \underset{\Psi_{n-1}}{\operatorname{argmax}}\, p(x_n|\Psi_n)p(\Psi_n|\Psi_{n-1})\mu_{n-1}(\Psi_{n-1})$$

One therefore obtains a matrix of paths $\Big(\Gamma(\Psi, n)\Big)$.

When this is done, one has to calculate the most likely path. This is done by backward recursion by starting from the most likely ending point and then just going back on the path we calculated previously:

$$\boxed{\begin{cases} \Psi_T^* = \underset{\Psi_T}{\operatorname{argmax}}\, \mu_T(\Psi_T) \\ \\ \Psi_n^* = \quad \Gamma(\Psi_{n+1}^*, n+1) \quad \text{for } n = T-1, \ldots, 1 \end{cases}}$$

This code has been implemented. As usual, log-probabilities are used in calculations. The results yielded by this approach are compared with the ones from the forward-backward one.

3.3 Testing of the Viterbi algorithm

Again before applying the algorithm to decode the barcode, we check that indeed the plots of s_n, c_n and m_n are coherent (fig. 6).

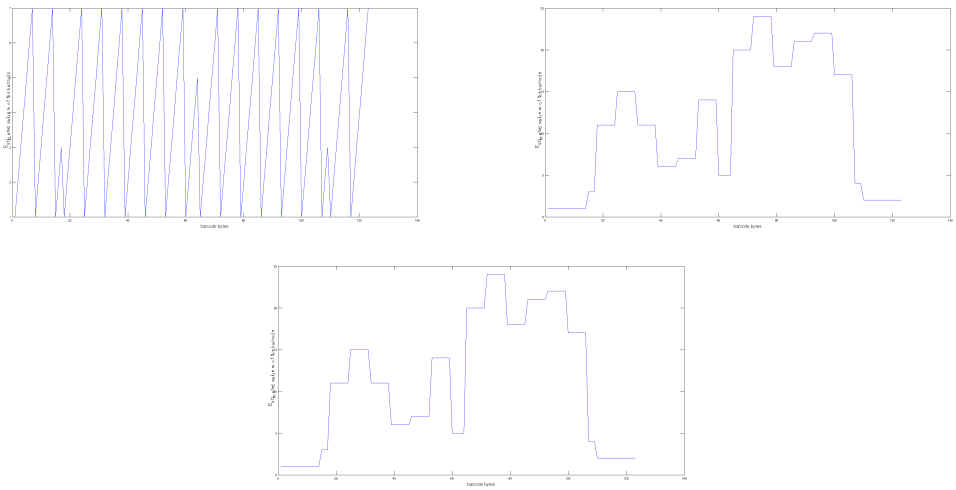


Figure 6: Plot of the three latent variables in function of pixel number n

As one can see, the results are identical to the ones of fig. 4: it means that poth of our algorithms are very likely correct. We now proceed to the barcode decoding task.

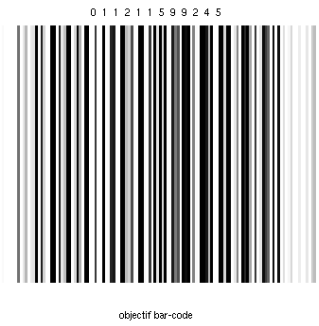
3.4 Decoding the barcode

We first try to decode the basic code 012345678905 with $\mu_0 = 20$, $\mu_1 = 250$ and $\sigma^2 = 5$, i.e. with low noise. As one can see by running the code for such parameters, the algorithm has no trouble to do so. Similarly for other random generated codes we obtain similar results.

When increasing the noise to $\sigma^2 = 60$, the scanline still achieves to decode the barcode.

```
Real code:      012345678905
Decoded code:   012345678905
```

Figure 7: Dummy barcode in an almost noiseless case ($\sigma^2 = 5$)




```
Real code:      011211599245
Decoded code:   011211599245
```

Figure 8: Randomly generated barcode with $\sigma^2 = 60$

However when the noise becomes too important (e.g. $\sigma^2 = 200$) the scanline does not achieve to decode the code correctly anymore: the picture has been too corrupted.

This shows the limit of our code, although it is debatable if any other method would achieve to find the right code for such a corrupted barcode.


```
Real code:      852729661020
Decoded code:   552740030749
```

Figure 9: Randomly generated barcode with $\sigma^2 = 200$

4 Experimental results

Question 8 - In what follows, we are showing the results on the basis of the S state as it's the most visual plot we found reliable to transcript our results.

4.1 No Noise case

We show below a case of a randomly generated barcode with no noise $\sigma = 0$. However, we fix the variance of our observation law to $\sigma_0 = \sigma_1 = 1$.

We feature the smoothing distribution, the filtering distribution along with the most likely and the actual path.

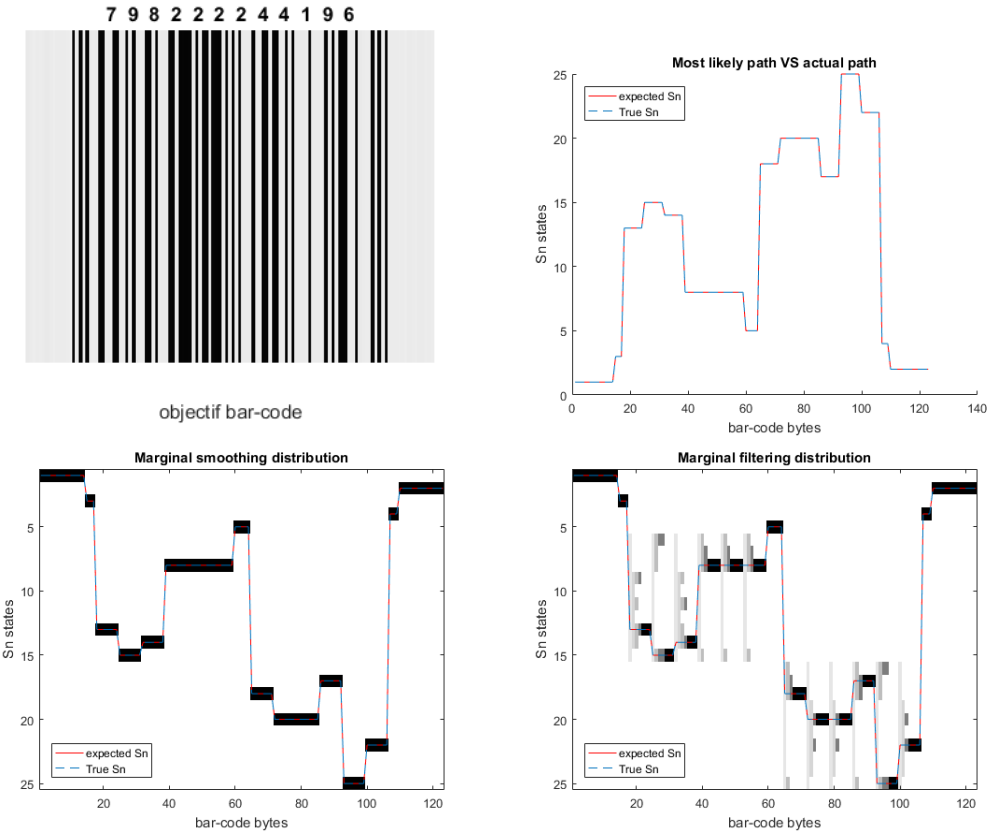


Figure 10: a no noise case

We observe here our ability to infer the actual barcode accurately. The behavior of both the smoothing and the filtering distribution is expected. In fact, the smoothing distribution has no error margin as we are using the full encoding of each character before making decisions. On the other hand, the filtering distribution starts with a uniform distribution over all possible states (as all encodings start with a 0, and as we have a uniform transition probability over possible states)

These results hold for all our simulation runs with no exception.

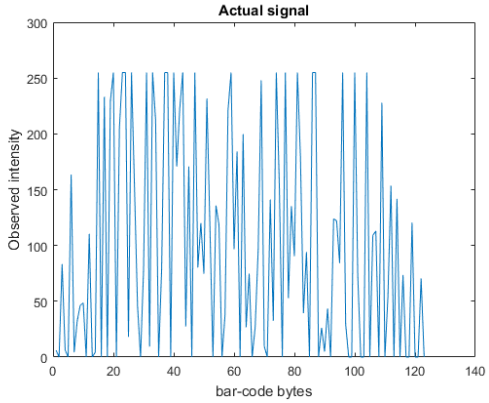
4.2 Noise case

In this section we are going to study three possible outcome of our parametrization of the HMM model.

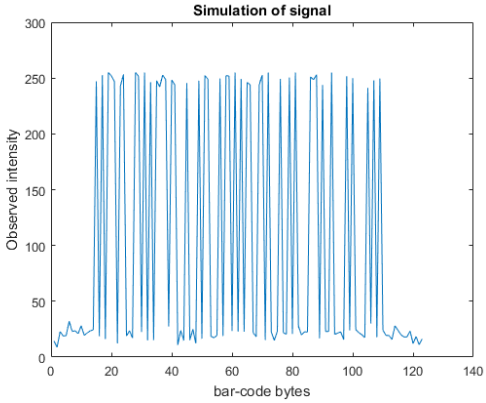
- We model poorly a reasonable noise $\sigma_0 = \sigma_1 >> \sigma$ or $\sigma_0 = \sigma_1 << \sigma$
- We model accurately a reasonable noise $\sigma_0 = \sigma_1 = \sigma$
- We model accurately a an unreasonable noise $\sigma_0 = \sigma_1 = \sigma > \sigma_{thresh}$

4.2.1 Bad prior

Something that would give us a good hint of what is going to happen is comparing the observation that we have and a simulation of our HMM model.



(a) $\sigma_0^2 = \sigma_1^2 = 100$



(b) $\sigma_0^2 = 20$

The problem is in fact of computational nature. In fact as we expect little noise, we end up with zero likelihood in the tail of our Gaussian, leading to numerical undefinability. which leads to no estimation at all as shown bellow.

The problem holds when we over estimate the noise, however in this case the issue is due to high unrestrictive model that leads to nondeterminism in some case. The likelihood is going to be similar.

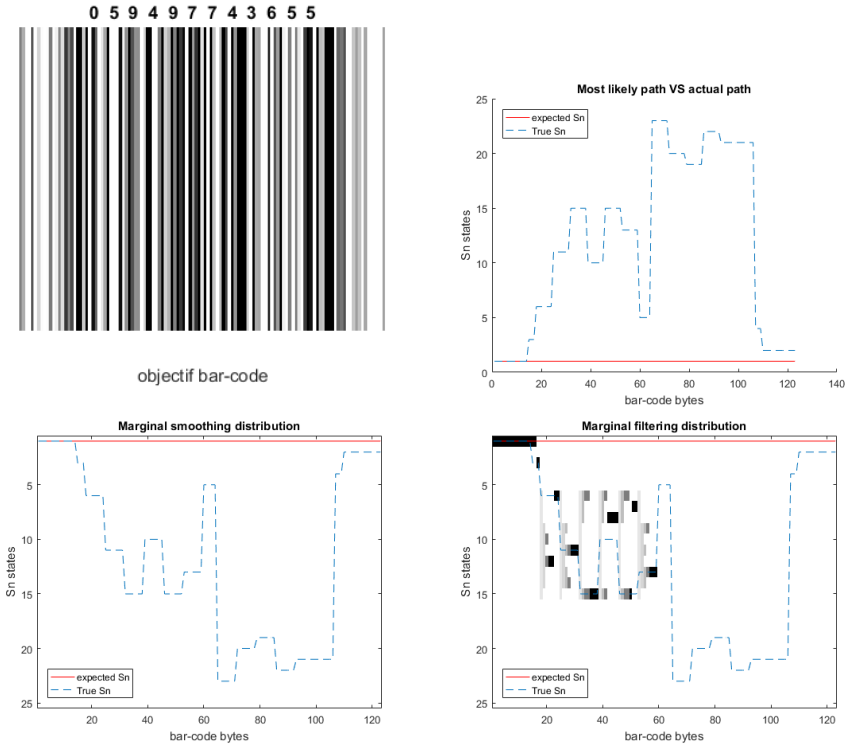


Figure 12: An under estimation of the noise

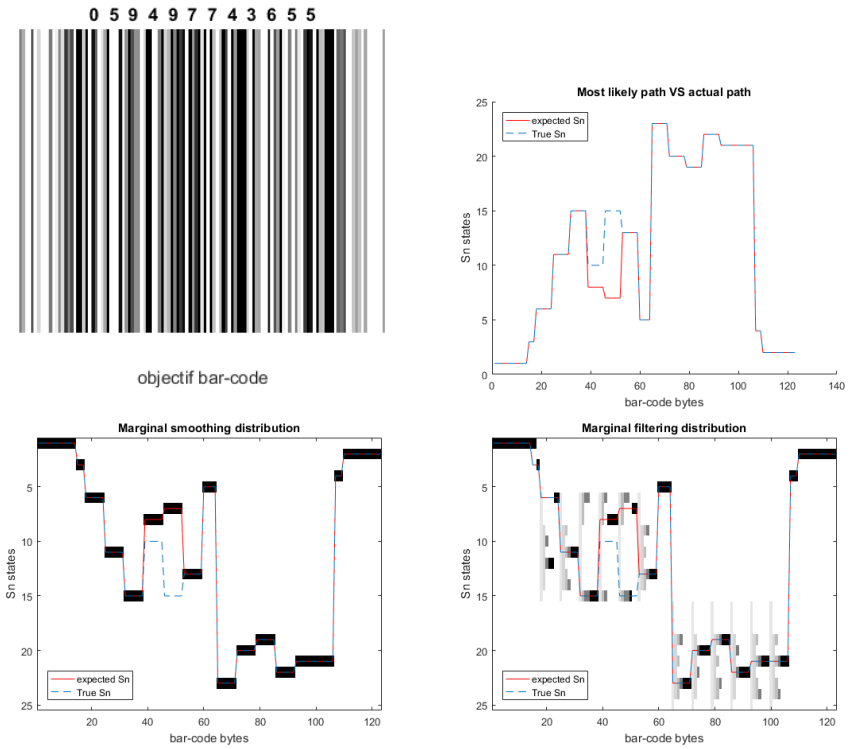


Figure 13: An over estimation of the noise

4.2.2 Good prior

In what follows we are going to work under the same previous noise, however we are going to use a good estimation of this noise.

Even thought the smoothing distribution doesn't show disturbances, we are sure about our decision in the filtering distribution only starting a quite late number of bytes compared to the low/no noise case.

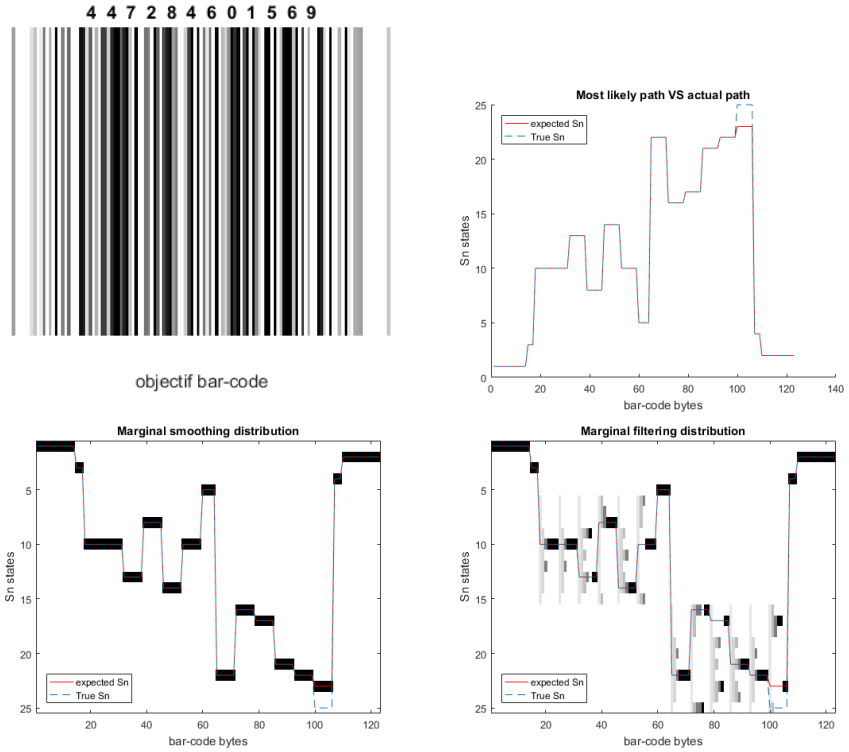


Figure 14: a moderate noise case under good estimation of the noise

4.2.3 Noise threshold

Even though with the HMM model we are able to infer the bar code under reasonable noise, we end up with the limit of our model once the noise becomes excessive.

In fact, if the noise become high such that we are likely to generate a black strip when we are supposed to observe a white one, the information contained in the Xn becomes insufficient to make any prediction. Which leads to noisy distribution and unreliable predictions.

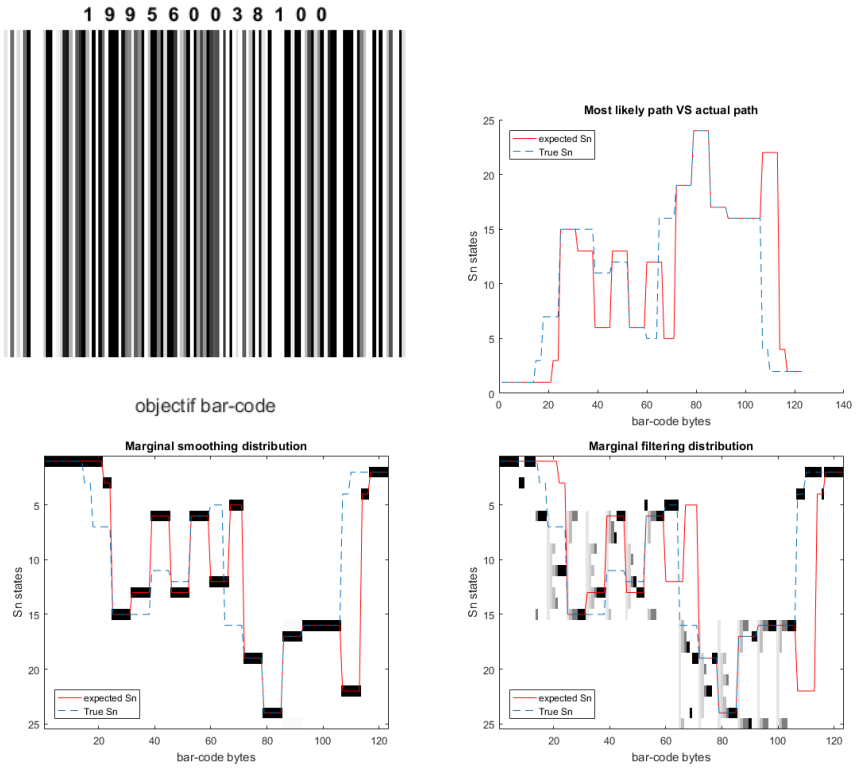


Figure 15: An overly high noise case under good estimation of the noise

4.3 Other marginal smoothing and filtering distributions

One could also be interested in the smoothing and filtering distributions of c_n and m_n. However those present a less interesting behavior than s_n since they are "easier to predict", as one can see with fig. 16.

As previously shown for the marginals of s_n, the smoothing distribution perfectly predicts the value of both m_n and c_n, and presents an improvement over the filtering distribution alone.

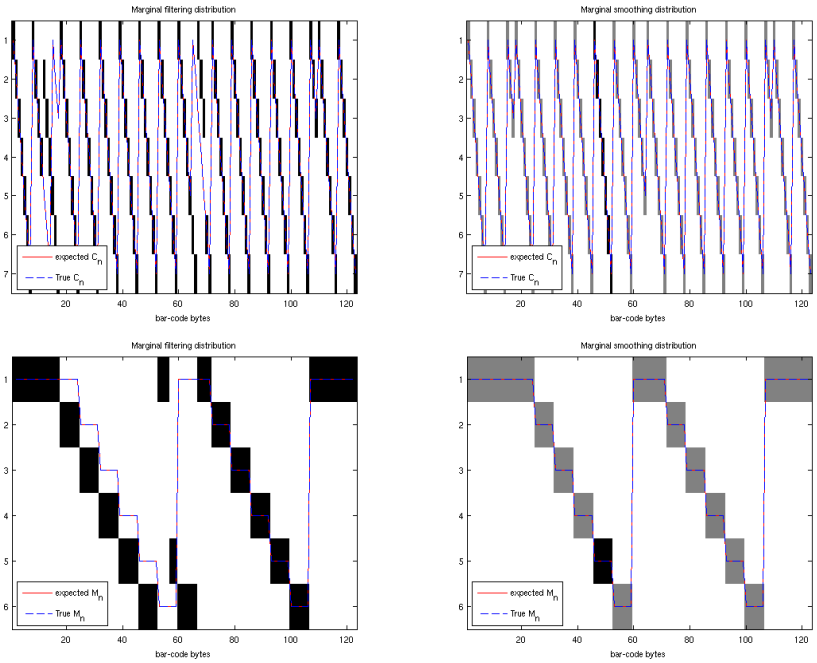


Figure 16: Plot of the marginal smoothing and filtering distributions of c_n (top) and m_n (bottom)

5 Conclusion

In this project, a probabilistic approach has been considered to reconstruct barcodes from noisy observations, more precisely through the modelization of the situation by a hidden Markov model. Two approaches have been implemented with both allowing to efficiently reconstruct the code, namely the Forward-Backward algorithm and the Viterbi one. For moderate amounts of noise, both achieve to perfectly decode the image and yield the expected result. It is only for very high amount of noise that they fail to do so.

Therefore the developed code could probably be implemented in a real life situation to optimize the speed and reliability of scanning in many commercial fields.