Stochastic machines dedicated to Bayesian inference for source localization and separation

Raphael Frisch

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Raphael Frisch

Thèse dirigée par **Laurent Fesquet**
et codirigée par **Emmanuel Mazer**

préparée au sein **Laboratoire Informatique de Grenoble**
et de **École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Machines stochastiques dédiées à l'inférence Bayésienne pour la localisation et séparation de sources

Stochastic machines dedicated to Bayesian inference for source localization and separation

Thèse soutenue publiquement le **14 novembre 2019**,
devant le jury composé de :

Madame, Marie-Christine Rousset
Professeur, Université Grenoble Alpes, Présidente
Monsieur, Olivier Sentieys
Professeur, Université de Rennes 1, Rapporteur
Monsieur, Sylvain Marchand
Professeur, Université de La Rochelle, Rapporteur
Monsieur, Eric Jonas
Assistant Professeur, Université de Chicago, Examinateur
Monsieur, Laurent Fesquet
Maître de conférences, Institut Polytechnique de Grenoble, Directeur de thèse
Monsieur, Emmanuel Mazer
Directeur de recherche, CNRS, Co-Encadrant de thèse, Invité

Résumé

L'ordinateur est sans aucun doute l'une des inventions les plus importantes du siècle dernier, dont l'impact ne peut être surestimé. Au fil des années, ils sont devenus de plus en plus puissants grâce à l'optimisation constante des processeurs. Avec un besoin croissant en puissance de calcul, et notamment à cause de l'IA, les processeurs sont devenus plus rapides que jamais. Cependant, à cause des limites physiques, la loi de Moore touche à sa fin. Par conséquent, il est nécessaire de proposer des alternatives. C'est le but de la communauté *rebooting computing*. Dans ce travail, nous nous proposons d'utiliser le calcul stochastique pour construire des architectures dédiées à l'inférence bayésienne visant une faible consommation d'énergie. Nous avons développé deux machines, à savoir la *Bayesian machine* (BM) et la *Bayesian sampling machine* (BSM). Dans cette thèse, nous nous intéresserons à deux applications de traitement du signal : la localisation de sources sonores (SSL) et la séparation de source. Pour la SSL, nous présentons trois méthodes utilisant la *Bayesian machine*. La première méthode fonctionne dans le domaine temps-fréquence, nécessitant le calcul de la transformée de Fourier. La deuxième est entièrement dans le domaine temporel. La troisième approche est une méthode de localisation multi-sources qui est basée sur la seconde. De plus, nous proposons une technique permettant d'accélérer le calcul stochastique d'un facteur 10^3. Nous avons également développé une méthode de calcul des vraisemblances afin de réduire la mémoire de notre machine. Nous avons simulé les trois méthodes et fait des expérimentations en environnement réel. Nous présentons la consommation d'énergie obtenue via des simulations ASIC. Pour la seconde application, la séparation de source, nous introduisons une machine plus générale, la *Bayesian sampling machine*, qui est basée sur l'échantillonnage de Gibbs. Nous présentons une méthode basée sur l'échantillonnage pour séparer des sources sonores. Cette méthode a été validée en simulation.

Abstract

Computers are without doubt one of the most important invention of the last century, whose impact cannot be overestimated. Over the years they became powerful, due to the constant optimization of their processors. With the growing need of computing power due to AI, processors have become faster than ever. However, since we are reaching the power wall, Moore's law is coming to an end. Therefore, a young research community called *rebooting computing* is looking for alternative computation architectures. In this work, we propose to use stochastic computing to build architectures dedicated to Bayesian inference aiming low-power consumption. We develop two machines, namely the *Bayesian machine* (BM) and the *Bayesian sampling machine* (BSM). In this thesis, we look at two signal processing applications: Sound Source Localization (SSL) and Source Separation (SS). For SSL, we introduce three methods using the BM. The first one is working in the time-frequency domain and hence uses the Fourier transform. The second one is running entirely in the temporal domain. The third one is a multi-source localization approach based on the previous method. We present a technique to speed up the stochastic computation by a factor of up to 10^3. Moreover, we designed an on-chip likelihoods computation mechanism to reduce the memory needs of our machine. Furthermore, we ran simulations and real world experiments to validate our methods. We made ASIC simulations to evaluate the power consumption. For the second problem, the source separation, we introduce a more general machine, the *Bayesian sampling machine*, which is based on the Gibbs sampling approach. We present a sampling method to solve source separation and run simulations to show the effectiveness of this technique.

Contents

1 Introduction **1**
1.1 Context . . . 2
1.2 Overview . . . 3
1.3 Thesis outline . . . 3
1.4 Contribution . . . 5

2 Related Work - rebooting computing **7**
2.1 Birth of rebooting computing . . . 7
2.2 Approximate computing . . . 9
2.3 Alternatives based on physical phenomena . . . 10
2.4 Neuromorphic computing . . . 11
2.5 Stochastic computing . . . 11
2.5.1 Accuracy . . . 12
2.5.2 Correlation . . . 13
2.5.3 Applications . . . 13
2.5.4 Improvements . . . 13
2.5.5 Random number generators . . . 14
2.6 Probabilistic programming . . . 14
2.7 Conclusion . . . 16

3 Mathematical foundations **17**
3.1 Bayesian programming . . . 17
3.1.1 Discrete variables . . . 17
3.1.2 Probability . . . 18
3.1.3 Normalization . . . 18
3.1.4 Conditional probability . . . 18
3.1.5 Variable conjunction . . . 19
3.1.6 Bayes theorem . . . 19
3.1.7 Marginalization rule . . . 19

3.1.8 Joint distribution 20
3.1.9 Decomposition 20
3.1.10 Parametric form 20
3.1.11 Inference 20
3.2 Exact inference 21
3.3 Approximate inference: Sampling algorithms using Markov chains 22
3.3.1 Markov chains 22
3.3.2 Perron-Frobenius theorem 24
3.3.3 Stationarity 25
3.3.4 Detailed balance 26
3.3.5 MCMC algorithms 26
3.3.6 Metropolis algorithm 27
3.3.7 Metropolis-Hastings algorithm 28
3.3.8 Connection between transition matrix and Markov chain 29
3.3.9 Gibbs sampler 29
3.4 Notations 30

4 Stochastic sampling machines for Bayesian inference 31
4.1 Stochastic computing 31
4.2 Bayesian machine 34
4.2.1 Bayesian fusion 35
4.2.2 Architecture of the machine 35
4.2.3 OP block 37
4.2.4 Tackling temporal dilution 38
4.2.5 Max-Normalization 39
4.2.6 Application: boat localization 41
4.3 Approximate inference - Bayesian sampling machine 44
4.3.1 Draw-gate 47
4.3.2 Application: boat localization with a large grid 48
4.4 Conclusion & discussion 50

5 Audio foundations 53
5.1 Sound Source Localization (SSL) 53
5.1.1 Task definition 54
5.1.2 Basics of audio processing 54
5.1.3 State of the art of source localization 58
5.2 Source separation 60
5.2.1 Task definition 60
5.2.2 State of the art of source separation 61

5.2.3 Independent Component Analysis 62
5.2.4 Binary Masking 63
5.2.5 Probabilistic Models for source separation 64
5.3 Discussion 66

6 Mono-Sound Source Localization in the time-frequency domain 69
6.1 Signal pre-processing 69
6.2 Probabilistic model 72
6.3 Bayesian machine adapted to SSL 74
6.3.1 Improvement of the Bayesian Machine: BM-sliced . . 76
6.4 Implementation on FPGA 77
6.4.1 Circuit design in VHDL 78
6.4.2 On-chip likelihood computation 80
6.5 Simulation environment 85
6.5.1 Simulation data flow 85
6.5.2 Sound pre-processing 86
6.5.3 BM simulation tools 87
6.5.4 Standard experimental setup 88
6.6 Simulated experiments 89
6.6.1 Localization performance 89
6.6.2 BM-standard vs. BM-sliced 90
6.6.3 Impact of the probability discretization 93
6.6.4 Impact of the sensor precision 95
6.6.5 Impact of the LFSR 96
6.6.6 Computation accuracy depending on the number of slices 96
6.6.7 Power consumption measurements 98
6.7 Real world experiments 101
6.7.1 Pre-processing on chip 101
6.7.2 Experimental set up 102
6.7.3 Results 104
6.7.4 Optimization - filtering 104
6.7.5 Optimization - bypass 104
6.7.6 Optimization - learned model 105
6.7.7 Circuit area 106
6.8 Conclusion 106

7 Mono-Source localization in the temporal domain 111
7.1 Probabilistic model with attenuation 111
7.1.1 Bayesian machine adapted to SSL 114
7.1.2 Simulations 114
7.1.3 VHDL implementation 117
7.1.4 Experimentations 119
7.2 Probabilistic model without attenuation 124
7.2.1 Modified setup 125
7.2.2 Localization map 126
7.3 Conclusion 126

8 Multi-Source localization in the temporal domain 129
8.1 Localization method 129
8.1.1 Probabilistic model for single-frame single-source SSL 130
8.1.2 Fusion of frame-wise results for multiple-source SSL . 131
8.2 Implementation on the Bayesian machine 132
8.3 Simulations 134
8.3.1 Setup 134
8.3.2 Localization performance 135
8.3.3 Robustness to various source locations 137
8.4 Conclusion 137

9 Source separation 139
9.1 General model 139
9.2 Sampling using Gibbs 141
9.3 Bayesian Sampling Machine adapted to perform the source separation 143
9.4 Simulations 144
9.4.1 Simplified sound simulator 144
9.4.2 Standard experimentation setup 145
9.4.3 Sampling space 146
9.4.4 Separation result 146
9.4.5 Separation result with noise 149
9.4.6 Separation result using only two microphones 152
9.5 Running the Bayesian sampling machine 154
9.5.1 Simulations on the modified Bayesian sampling machine 156
9.6 Conclusion 156

10 Conclusion & Discussion 159
10.1 Discussion 161

A Notations **163**
A.0.1 Specific notations for probabilistic inference 163
A.0.2 Specific notations for audio signal processing 164
A.0.3 Specific notations for the applications (SSL and source separation) 164

B LFSR issue **167**

C Proof draw gate **169**

List of Figures

2.1 Evolution of the CPU performance and clock speed between 1985 and 2015. Figures taken from [63]. 8
2.2 Illustration of the power limit. 8
2.3 The potential of dedicated hardware for more energy efficient. Each point represents one particular architecture. Each subfigure covers the period from 2009 to 2013. Taken from [34]. . 9

4.1 The multiplication of 3 probabilities p_1, p_2 and p_3 using stochastic bit streams. 32
4.2 Probability of a value depending on the length of a stochastic bit stream for a bit stream representing the value 0.8. In blue, a 5 bits bit stream. In orange, a 10 bits bit stream. In green, a 40 bits bit stream. 33
4.3 An example of a 16-bit Galois LFSR. 34
4.4 Architecture of the BM including a zoom on a single OP block of the machine. Bit streams are represented by red arrows. Blue arrows illustrate fixed-point numerical values. 36
4.5 Detailed overview of the Bayesian machine architecture including the counters. 37
4.6 Example of an SBG using a 16-bit Galois LFSR. 38
4.7 Example of a max-normalization on a column of 5 lines. . . . 40
4.8 Setup for the boat localization example with a 16 by 16 grid. 42
4.9 Distribution of the different sensors which are fused by the Bayesian machine. 44
4.10 Computed distribution for different lengths of bit streams. . . 45
4.11 Architecture of the Bayesian sampling machine. 46
4.12 Setup for the boat localization example with a 1024 by 1024 grid. 49

4.13 Plot of the likelihood distributions for the different sensors. In blue the distance sensors. In orange the bearing sensors. . 51
4.14 Plot of the different explored positions to localize the boat placed in (512,512). 52

5.1 The mixing process of two sources recorded by two microphones. 60
5.2 Flow of the binary masking method. Figure taken from [13]. . 64
5.3 NMF example for xylophone notes. Figure taken from [97]. . 65

6.1 Schema of the source-to-microphones wave propagation. . . . 71
6.2 Architecture of the Standard-BM including a zoom on a single OP block of the machine. Bit streams are represented by red arrows. Blue arrows illustrate fixed-point numerical values. . 75
6.3 Architecture of a BM-sliced with two slices of $q = n/2$ columns each and 1 re-sampling unit in between. 76
6.4 Re-sampling unit showing and example of max-normalization with the line 2 being at the maximum. 77
6.5 Filter-like implementation of the machine for the VHDL design. 78
6.6 The simplified architecture of the VHDL implementation of the BM-sliced for the SSL. 80
6.7 The automata of the global FSM of the BM. 81
6.8 The OP block before introducing the on-chip normalization processing with the memory block containing an entry for each sensor value. 82
6.9 The mechanism used for the on-chip computation of the likelihood value for each OP block. 82
6.10 The detailed architecture of the VHDL implementation of the BM-sliced for the SSL. 83
6.11 The principle of the pipelining technique used to speed up the initialization phase. 85
6.12 Simulation data flow showing the different simulation possibilities. 86
6.13 Setup of the room simulated to generate the sound for our experiments. 88
6.14 Probability map for BM-sliced at 5.000.000 steps. Maximum points of Ground truth source position (red), Source position estimated by the BM-sliced (green) which stays stable after 10.000 steps, Source position estimated by the EM algorithm of [40] (orange), Source position estimated by exact inference method (blue). 91

6.15 Sum of all output counters (on a log-scale) as a function of the number of computation steps. 92
6.16 Comparison of the Kullback-Leibler divergence between the distribution computed by the BM (standard and sliced version) and the exact distribution as a function of the number of computation steps. 93
6.17 KLD as a function of the re-sampling threshold value for different *n_bit*. *n00p04t* being the exact inference. 94
6.18 KLD as a function of the re-sampling threshold value for different discretization of the sensor data. 95
6.19 KLD as a function of the re-sampling threshold value for runs with LFSR as random number generators (t) and with the random number generator of C++ (f). 97
6.20 KLD as a function of the re-sampling threshold value for different number of columns per slice. 98
6.21 Overview of the different modules present in the global system and their connections. 102
6.22 Phase difference computation module for 1 microphone pair. . 102
6.23 Sound acquisition set up. 103
6.24 Microphone set up in the room. 103
6.25 Localization map obtained using the free field model. 104
6.26 Evolution of the localization map after filtering over 4 frames. 105
6.27 Examples of localization when using no threshold (NT), a medium threshold (MT) and a high threshold (HT) using the filtering method (F) or without filtering (no filtering - NT). . 108
6.28 Localization map using the learning method at position 54. . 109
6.29 Comparison between the original free field model and the learned model at position 0. 109
6.30 Comparison between the original free field model and the learned model at position 7. 109
6.31 Comparison between the original free field model and the learned model at position 36. 110
6.32 Comparison between the original free field model and the learned model at position 56. 110
6.33 Comparison between the original free field model and the learned model at position 63. 110

7.1 Architecture of the Bayesian machine used for the temporal localization method. 114

7.2 Setup of the room simulated to generate the sound for our experiments. 115
7.3 Localization map with for a frame of 30 samples. 117
7.4 Different microphones used for the hardware implementation of the SSL method. 118
7.5 Preprocessing of the likelihood on-chip for the temporal localization method. 119
7.6 Examples of localization when using different size of frames (recording time T). 121
7.7 Examples of localization when adding noise to the recorded signal. 122
7.8 Microphone signals obtained in a real world recording. 122
7.9 Setup of the room simulated to generate the sound for our experiments for the model without attenuation. 125
7.10 Localization maps provided by method using temporal comparison without attenuation model. 126

8.1 Simplified representation of the Bayesian machine architecture used to compute the inference equation for SSL and the fusion. 133
8.2 Simulated room setup for our multi-source localization method. 134
8.3 (a) to (c): Posterior distribution map obtained for 3 very short time-frames of a given 50-frame bloc. (d) Final distribution map after fusion over 50 frames. The two black squares correspond to the actual positions of the two sources. 136

9.1 Architecture of the Bayesian sampling machine (BSM) adapted to the source separation problem. 143
9.2 Position of the speakers and the microphones to simulate the sound for our experiments. 145
9.3 Visualization of the sampling space. 146
9.4 Signal recorded by microphone 1. 147
9.5 Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps. 147
9.6 Reconstructed signal of source 1 after sweep 0. 148
9.7 Reconstructed signal of source 1 after sweep 1. 148
9.8 Reconstructed signal of source 1 after sweep 7. 148
9.9 Quality of the estimated result: original signal in red, estimated signal in green and blue the difference between the original and the estimated signal. 149
9.10 Signal recorded by microphone 1 with added Gaussian noise. 150

9.11 Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps when adding Gaussian noise to the recorded signal. 150
9.12 Reconstructed signal of source 1 after sweep 15. 151
9.13 Quality of the result: original signal in red, estimated signal in green and blue the difference between the original and the estimated signal. 151
9.14 Spectrogram of the estimated signal for source 1. 152
9.15 Position of the speakers and the two microphones. 152
9.16 Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps. 153
9.17 Reconstructed signal of source 1 after sweep 6. 153
9.18 The four Gaussian distributions of the likelihoods separated in two main groups. 155
9.19 The four Gaussian distributions of the likelihoods superposed. 155
9.20 Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps. 157
9.21 Reconstructed signal of source 1 after sweep 20. 157

B.1 Example of bit streams generated using and 32 bits LFSR. . 168

List of Tables

4.1 Output of the draw gate in case of $w = 3$. Each line of the table is associated to a 3-bit input to the draw-gate at time t. The output is given depending on the previous output of the draw-gate X_{t-1} which is associated to the columns of the table. 48

6.1 Results of the ASIC simulations for both architectures: the original and the optimized version. 100
6.2 Circuit area measurement of the system mapped on the FPGA. 106

7.1 Circuit area measurement of the system mapped on the FPGA for the SSL without Fourier transform. 123
7.2 Circuit area measurement of the system mapped on the FPGA for the SSL without Fourier transform (in red) compared to the SSL with Fourier transform (in blue). 124

Chapter 1

Introduction

Computers are without doubt a most important invention of the last century, whose impact cannot be overestimated. The evolution of their computing power has been awesome and the applications now running on them are astonishing. Processors, which are their heart, have been optimized again and again, allowing their resulting performances to improve literally every year. However, one can ask whether this purely "brute force" evolution is the only way forward, and, first of all, whether it can go on forever.

Over the years, the race of providing more and more computing power has pushed the researchers to make choices on hardware specific concepts. This resulted in some concepts to be neglected. One of these technologies which has been less investigated in the past is *stochastic computing.* Now that the Moore's law[1] has come to an end, one can feel the need for some alternative computing schemes. Preparing the post Von Neumann era, *rebooting computing* is a research movement, which aims to develop unconventional computing schemes.

Stochastic computing allows to design low power circuits, which in the age of the Internet of Things (IoT) have promising advantages. The IoT will suffer from the communication network bottleneck as the expected amount of data transmitted to the cloud will increase drastically. Therefore it is crucial to process the data on-site close to the sensor to reduce the transmitted data. We will move from sensor data processing to systems, which incorporate enough intelligence to make adequate decisions. This forms the concept of Artificial Intelligence of Things (AIoT) which combines Artificial Intelligence (AI) with the Internet of Things (IoT) to accomplish more efficient IoT operations.

[1]Moore's law is the observation that the computing power doubles every 18 months.

For IoT, enormous amounts of sensor data must be processed. Sensors, which are the interface to the real world can provide faulty data. Especially when using low budget sensors, the amount of inaccurate data increases. Therefore, it is imperative to use robust models to process the data. *Bayesian models* are robust to faulty sensor data. Using Bayesian programming constitutes a strong foundation to develop a tolerant system. We mix Bayesian programming with stochastic computing to realize an alternative computing approach.

1.1 Context

The use of non-standard architectures to process uncertain information has experienced a remarkable growth in recent years, see for example the MIT Probabilistic Computing Project [5] and the DARPA projects PPAML [4] and UPSIDE [6]. A European counterpart is the European project BAMBI [1] (Bottom-up Approaches to Machines dedicated to Bayesian Inference) during which we proposed some first machines dedicated to Bayesian inference. The Bambi project revealed the potential of stochastic computing when considered at the bit level.

The work presented in this thesis is part of the MicroBayes project [3]. The project consortium consists of experts of different disciplines: mathematics, computer science, signal processing, micro-electronics.

The development of modern computers is mainly based on the increase of performances and the decrease of area and energy consumption. This incremental evolution is notable, but it involves no notable modification of the basic principles of computation. In particular, all the components perform deterministic and exact operations on sets of binary signals. These constraints obviously impede further sizable progresses in terms of speed, miniaturization and power consumption. The goal of the MicroBayes project is twofold:

- to investigate a different approach to perform computations, namely stochastic computing using stochastic bit streams.
- to show that stochastic architectures can outperform standard computers to solve complex inference problems both in terms of execution speed and of power consumption.

In this thesis, we demonstrate the interest and feasibility of stochastic computing on two applications involving low-level information processing from sensor signals, namely sound source localization and separation. This radical change of current computation models, at their deepest design level, may very well lead to the elaboration of low power reactive systems directly connected to their environment through noisy sensors.

1.2 Overview

Two architectures dedicated to Bayesian inference have been developed in this thesis. As they are simple, it makes them ideal candidates for low power architectures. Since they are based on stochastic computing, they are robust. Due to this simple design, the circuit area is small, hence they are cheap to produce. In this thesis, we go beyond the conceptual development and show how to solve typical signal processing problems using our Bayesian machines. Our solutions detail the structure and includes simulations and hardware design and implementation.

The first architecture, the *Bayesian machine* is devoted to inference problems having a reasonably small search space. Typically, it can be used for sensor fusion applications. This architecture has been utilized for a classical signal processing task: sound source localization. Several approaches under specific assumptions have been proposed. All methods have been tested in simulation and some in more realistic environments. These experimentations lead to a first realization of an ASIC prototype, currently under development.

A second machine, the *Bayesian sampling machine* has also been designed and is dedicated to more complex and general inference problems having, larger sampling spaces. It is based on sampling methods such as the Gibbs sampler. Source separation, which is the second application in our project, has been successfully run on the Bayesian sampling machine. High level simulations and circuit level simulations have been performed to evaluate the performances of this machine. Note that for both signal processing applications, we do not aim to achieve the state of the art performance but the main objective of this work is to show the potential of Bayesian machines.

1.3 Thesis outline

This thesis is organized into three main parts: an overview of the related work (chapter 2 and 5) is followed by background knowledge (chapter 3 and chapter 5) to finally present our machines (chapter 4) adapted to the different applications (chapters 6 to 9). Since this thesis is multi-disciplinary, we aim to introduce all the necessary concepts to understand the contributions presented in this document. The expertises of this work range from computer science to electrical engineering and signal processing.

Chapter 2 presents the work related to the rebooting computing research community. With the growing need of alternative computing architectures, several approaches have been proposed. The main alternative computing schemes are reviewed in this chapter.

The background on Bayesian modelling is provided in chapter 3. We present Bayesian programming as well as the notion of exact inference. Also, approximate inference methods are explained. Sampling methods are at the heart of our second machine, namely the Bayesian sampling machine.

The central part of our project, the stochastic machines for Bayesian inference, called *Bayesian machines*, are shown in chapter 4. First, the stochastic computing approach is introduced. Both our machines are presented in this chapter including their components.

Since our project aims to run signal processing applications on our machines, we need to introduce the required basic concepts of this thesis in signal processing. The presentation of the signal processing background is provided in chapter 5. The applications are sound source localization and source separation. Besides defining the tasks, we also review the approaches that have been proposed over the years by the signal processing community.

Chapter 6 is the first chapter presenting a Bayesian machine that deals with the sound source localization problem. The approach described in this chapter is based on the Fourier transform. The machine and its optimizations are presented. Some simulation results and real world experiments are demonstrated. Finally, the circuit of the machine is simulated to obtain its power consumption and circuit area.

Since a large part of the electrical circuit of the approach from the previous chapter is dedicated to the Fourier transform, we propose a more lightweight solution in chapter 7. This method works in the temporal domain to reduce the circuit area. Simulated experiments are provided.

Using the method of chapter 7 and enhancing the system, we are able to achieve multi-source localization. The approach is presented in detail in chapter 8. The modifications of the machine are stated. Simulations have been run and the results are shown.

Finally, dealing with the second application, chapter 9 presents the work done on source separation. For this purpose we introduce the second machine, namely the Bayesian sampling machine is introduced as well as its adaptation to the application. A large number of simulations have been conducted to determine the strengths and limitations of the proposed system.

Finally, chapter 10 concludes on the presented work. The different proposed methods are summarized and some future work is proposed.

Due to the different aspects of the work, our aim has been to deliver a manuscript as complete as possible to provide all the background knowledge required to understand our contributions.

1.4 Contribution

The goal of my thesis is the development of Bayesian machines, using Bayesian modeling and stochastic computing to solve Bayesian inference problems. Within the Microbayes project, Bayesian machines mainly were a theoretical construct, my work addresses real world applications to show the potential of these architectures.

I dealt with two signal processing applications, namely the sound source localization and the source separation. For each problem, I designed a dedicated Bayesian model and adapted the hardware to that task. I validated the proposed model on a high level software simulation before running simulations at circuit level. Moreover, I run experiments in a real world setup to show the performance of the system. Lastly, I implemented the circuit in VHDL to run it on FPGA. I also performed ASIC simulations to obtain a realistic evaluation of the power consumption. Currently, a prototype is under development to produce an ASIC.

Although stochastic computing has many strengths, the temporal dilution problem still remains present. I addressed this problem and provided an optimization to speed up the computation time and hence reduce the required energy by the machine. Moreover, the memory needs of the initial machine were tremendous. I proposed an on-chip likelihood computation method to decrease the memory in the circuit, which is the main power consumer. Furthermore, I designed a more generic machine dedicated to more complex inference problems, which I used for source separation.

My work resulted in four peer-reviewed conference articles, which I published in [46], [47], [48] and [55]. I also co-organized an IEEE workshop dedicated to "Emerging technology for probabilistic inference" at the 2019 IEEE International Nanodevices & Computing Conference in Grenoble where is presented my latest results.

Chapter 2

Related Work - rebooting computing

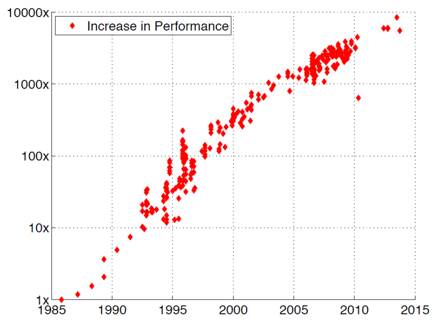
As the aim of this thesis is to develop alternative computing architectures, we dedicate this chapter to provide an overview of the different computing approaches that have been proposed over the years. Note that the state of the work of the signal processing part of this work is provided in chapter 5. First, let us understand why there is a need of new computing approaches.

2.1 Birth of rebooting computing

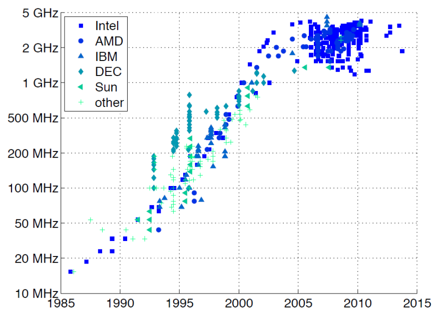
Computing has tremendously evolved over the years. Both, needs and performance keep increasing. Due to the effort of the research community, we are now able to design machines which are smaller, more efficient and less power consuming.

Many predictions have been made about the evolution of processor speed and especially Moore's law from 1965 made CPU cheaper over time. It stated that the number of transistors per chip are going to double every 18 months at constant cost [88]. Moreover, Dennard in 1974 predicted, based on physical observations, that the processors will become faster and more energy efficient [39].

In figure 2.1a one can see an overview of the performance of the CMOS processors of the last 30 years. One can see the performance increasing by nearly four order of magnitude. This picture has been taken from [63]. As the performance of a CPU is depending on the clock frequency, it increased as well, as shown in figure 2.1b. These figures were made possible thanks to the CPU database from Stanford [35] which reviewed all available CPUs



(a) Increase in performance.



(b) Evolution of clock frequency.

Figure 2.1: Evolution of the CPU performance and clock speed between 1985 and 2015. Figures taken from [63].

from Intel's first single-chip microprocessor Intel 4004 in 1971 to now.

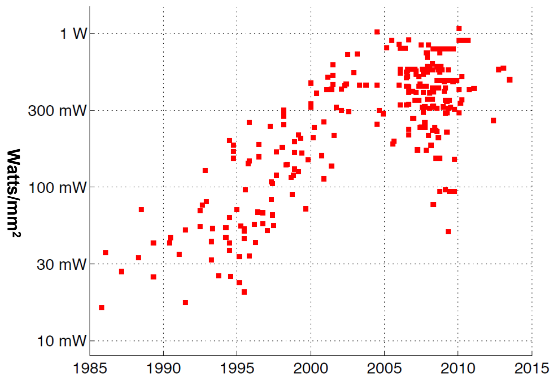


Figure 2.2: Illustration of the power limit.

However, because of physics, this increase can not continue forever. In fact, since 2005 the clock speed of the processors stopped increasing, as shown in figure 2.1b. The reason is the power density, which also increased along the years, illustrated in figure 2.2. Due to physical reasons, in 2005 we hit the so called *power wall* as it is physically not possible to augment the processor frequency because it generates too much heat that has to be dissipated. The energy required for cooling becomes too important and makes it unviable.

Over the years different techniques have been developed to increase the energy efficiency in hardware. The Graphics Processing Unit (GPU),

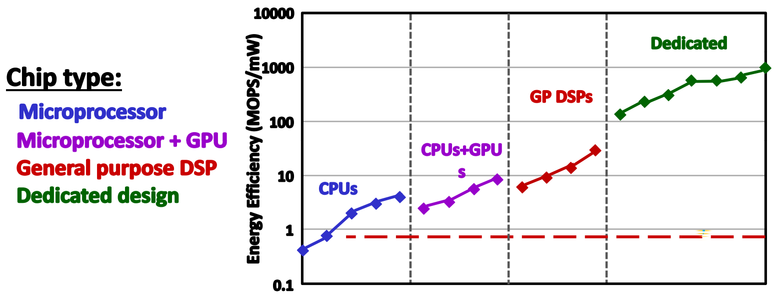


Figure 2.3: The potential of dedicated hardware for more energy efficient. Each point represents one particular architecture. Each sub-figure covers the period from 2009 to 2013. Taken from [34].

which have originally been designed for image and video processing, became heavily used for deep learning. The counterpart in signal processing is the Digital Signal Processor (DSP), which is an optimized micro-processor for signal processing applications. All this new hardware allowed to become more efficient in specific applications. The trends for application specific and dedicated hardware is getting further and ameliorates the efficiency, as illustrated in figure 2.3.

When looking at the computing energy problem explained in this section, one clearly sees the motivation of the development of new computing paradigms. This gave birth to the *rebooting computing* community, which aims to propose and design alternatives to the existing CPUs. Many different approaches have been presented. In the upcoming sections, we provide an overview of selected methods. A very simple idea is to study the impact on the energy consumption when reducing the accuracy of the system. Some investigate a totally unusual approaches based on new physical phenomena. A part of the community is basing its research on the human brain and focus on how it can be imitated. Moreover, stochastic computing, which uses an alternative representation of numbers is reviewed in detail. It can be seen as an human inspired approach. Finally, we look at the probabilistic programming community, which is working on software and hardware solutions.

2.2 Approximate computing

In recent years, a straightforward approach to reduce computing power called *approximate computing* has emerged [124]. It is also called *good enough computing* [116]. By taking conventional architectures and reducing certain parameters, one could save energy. In some scenarios, the accuracy of the results computed by conventional computing methods can be reduced. The goal is to find an appropriate trade-off between the level of accuracy of the application and the precision of the architecture to accomplish certain optimizations [85]. By accepting a drop of accuracy, one can save resources. Applications of approximate computing reach from video processing and signal processing to machine learning [124]. There are different methods to achieve power reduction. One can reduce the precision of the hardware by reducing the data representation (i.e. fixed-point arithmetic) in the system [84]. Moreover, approximated operators have been proposed [78] and [73]. Loop perforation is an example of what can be done on the software side. One can also scale the supply voltage which results in a quadratic reduction of the energy consumption. However, such techniques, i.e. voltage over-scaling, introduce timing errors. Dedicated hardware addressing this issue has been proposed in [101]. A detailed study of the impact of the fixed-point arithmetic and approximate operators has been presented in [15].

2.3 Alternatives based on physical phenomena

Many researchers are working on exploiting physical phenomena to achieve more efficient computing. The most popular right now is quantum computing, which is based on quantum mechanics. The potential of this approach could overcome conventional computing methods [45]. This is the reason why many projects deal with this method and plenty of companies invest in it. The biggest existing quantum processors have been designed by IBM (50 qubits), Intel (49 qubits) and Google (72 qubits). In 2019 a first commercial circuit-based quantum computer has been presented [2]: System One from IBM Q.

Although quantum computers are getting more and more attention of the research community, some people prefer to use quantum mechanics to accelerate certain parts of conventional computing. Moreover, in conventional machine learning quantum computing could be used as an accelerator [25]. Beside quantum, optical computing represents another major option. The power of light, i.e. photons, is already identified by the community. For information processing electrons are complimentary to photons [112].

In machine learning, one can also use optics to speed up computing. For example, some specific part of the computing can be done using analog optic devices in order to get a faster computation. Random projections allow to reduce the dimensionality of certain problems. However, it requires a random matrix. Using light going through several layers of scattering material provides random matrices. This technique has been presented in [107] and is currently developed by the startup LightOn. A first Optical Processing Unit (OPU) has been built. This method provides a very large amount of random matrices at high speed.

2.4 Neuromorphic computing

Another computing approach, called neuromorphic computing aims to imitate the human brain and build some dedicated hardware to achieve a more efficient computation. Some more on the neuroscience part in order to understand the human brain such as the BRAIN initiative or in [69]. Some research projects aim to build specific hardware such as the Spiking Neural Network Architecture, called SpiNNaker [49] of the Human Brain Project (HBP). Moreover, several companies recognized the potential of neuromorphic computing. Therefore, new dedicated chips have been developed. Some latest presented chips are TrueNorth from IBM [86] or the lately announced chips such as Loihi from Intel [36], the neuromorphic processor called Akida from BrainChip [65] and the Dynamic Neurormorphic Asynchronous Processor (DYNAP) from aiCTX [7].

Also, some interesting work, addressing the same application as we do, has been proposed in [110]. The goal is to imitate the barn owl's phase-locking on the Spikey chip [99]. The Spikey chip is an add-on for conventional computers plugged via USB to perform neuromorphic computing. In that work, the authors show that it is possible to achieve phase-locking on neuromorphic hardware, which provides an angular information with high azimuthal precision. This constitutes a first step towards sound source localization on neuromorphic hardware. Moreover, the usability of neuromorphic computing to be used for signal processing applications has been reviewed in [102].

Another approach close to neuromorphic computing is also human centered but more high level. It is called bio inspired computing. It does not only impact the computation but also the sensing and everything around the decision making.

2.5 Stochastic computing

The last computing approach we want to review is called *stochastic computing* (SC). We will have a more detailed look at it since the architectures we propose are based on it. It has originally been proposed by Gaines in 1969 in [51] and uses another data representation than conventional computing. Its Von Neumann, who introduced in 1956 the concept of using random processes in computers made of unreliable components and exploiting parallelism [94]. In stochastic computing, one represents the numbers as a bit stream of "0"s and "1"s, called stochastic bit streams. In theory, stochastic bit streams have an infinite length. However, as in practice they are finite, this introduces an uncertainty in the representation (cf. figure 4.2). Also, the stochastic bit streams can be seen as a neuron-like system [67]. It allows to perform robust and low-cost computing. Due to its specific data representation, it is very easy to implement more advanced operators such as multiplications [9]. For instance, the multiplication of two bit streams is performed by a logic AND gate. Moreover, as these circuits are very simple it represents a cheap alternative to conventional processors. SC is mainly used for applications where process variations and soft errors are not a big problem [100]. One main advantage of SC is to modify the precision of the computation since it is directly related to the length of the stochastic bit streams. This way, using very short stochastic bit streams allows to achieve fast (sometimes imprecise) computation [10]. This feature can, in certain domains, be very useful as shown in section 2.5.3.

2.5.1 Accuracy

Due to its data representation, SC allows to have significant power consumption reduction and build small circuits. However, it introduces inaccuracy in the computed result. The accuracy of the output of a stochastic system highly depends on two parameters. The first one is noise, which can typically be due to radiation or change of the supply voltage or even simple bit-flips. The second parameter is the number of stages in the system. A stage is one level of AND gates. When having several AND gates in a row, the system is a multi-stage system [87]. As the numbers encoded are between 0 and 1, when multiplying them, the result is a smaller value than the input. This results in a decease of the signal power (number of "1"s in the bit stream) and hence a decrease of the Signal-to-Noise Ratio (SNR). We call this phenomena the *temporal dilution* and addressed it in [46] (see chapter 6). In this situation, the system is more vulnerable to noise due to the small SNR. This problem of accuracy has been addressed and quantified [87]. In [93] the authors proposed a framework to manage accuracy in SC systems during the design phase.

2.5.2 Correlation

In order to obtain accurate results, the input bit streams have to be independent and uncorrelated. This comes with the cost of having separated and dedicated random number generators for each bit stream. Therefore the conversion of the binary numbers to the stochastic bit streams represents a significant portion of a stochastic circuit. In order to reduce the circuit area and to have a more scalable technology, researchers reviewed the basic principles of stochastic computing [8]. However, for certain operations it is possible to take advantage of correlation by sharing the random number generator among the different input streams. Therefore, the number of random number generators is reduced. However, this requires to review the basic stochastic operators [23]. In case of an image processing application, the circuit area has been reduced up to 90% [23].

2.5.3 Applications

The application field of stochastic computing is vast. Historically, its use was more focused on control or machine learning applications. In recent years, it has been used for embedded encoding/decoding hardware or in image processing for edge detection. In the following, we aim to provide a brief overview of some applications. For example, in image processing dedicated stochastic hardware for edge detection using the Robert cross operator has been described in [11]. Another approach for edge detection including its orientation has been presented in [23]. In [14], an implementation of a Deep Neural Network (DNN) has been proposed using SC. Here, compared to conventional architectures a 21% reduction in energy consumption has been achieved according to the authors. Moreover, SC as been used in telecommunication applications for low-density parity check (LDPC) decoding such as presented in [60] and [91]. Finally, in [87] the described system has been tested as Discrete Cosine Transform (DCT) block as part of a JPEG encoder.

2.5.4 Improvements

An asynchronous SC approach has been proposed in [56]. The idea is to remove the clock and design an asynchronous circuit in order to reduce energy consumption. Asynchronous design is often used in low-power systems. It consists of removing the clock of the circuit which requires a lot of energy. The authors apply this method to an IOT sensor which needs to be low power consuming. However, their concept aims to do an end-to-end asynchronous design which is highly questionable as the transmitter requires to have the binary version of the stochastic bit streams. Hence the need to have a stream-to-binary conversion module which requires a clock.

Another way to speed up stochastic computing is to use optics. One could use the optical domain to overcome the intrinsic serial aspect of stochastic computing. Light has the physical property of having a low latency and high bandwidth. In [42], a stochastic circuit has been implemented in the light domain. They implemented a second order polynomial function and operated it at 1GHz. Their power consumption was around 20.1pJ laser consumption per computed stochastic bit.

2.5.5 Random number generators

Generating stochastic bit streams requires to have random number generators (RNG). The research area of RNGs is immense since they are used for many different applications. However, in SC the quality of the random numbers does not need to be cryptographic. Hence one can utilize more simple RNGs such as pseudo-RNGs. In hardware implementations, the most prominent example is the Linear Feedback Shift Registers (LFSR) which provides a good enough quality for SC is widely used [9].

Unfortunately, the energy required to run LFSRs is still very high. Therefore, a lot of effort is put into developing RNGs based on physical phenomena. For example, Magnetic Tunnel Junctions (MTJs) form a promising alternative for low-power RNGs [59]. Moreover, entropy extractors can also be used for massive random bit generation [30], [29] and [28]. Also, image sensing sensors have been proposed as RNGs [108]. Finally, quantum RNGs using coupled quantum dots for SC have been investigated [83]. All these efforts combined open new possibilities in designing truly stochastic computers.

2.6 Probabilistic programming

Nowadays a lot of artificial intelligence tasks are done by Deep Learning (DL) approaches using Neural Networks (NN). Beside its strengths and potential, DL has drawbacks which make a lot of people question its future. For example, DL needs a lot of data in the training phase and during the inference its decisions can not be explained due to its "black box" behavior [82]. Therefore other more flexible solutions have been investigated.

Probabilistic programming (PP) forms a encouraging alternative to DL since probabilistic models are able to deal with incomplete data or sensor uncertainty [66]. Many efforts are currently done to develop better and faster

PP languages. Hence, the list of new PP languages is growing. Some examples are Church [57], Venture [81] and its newer version Gen [33], Anglican [122], Pyro [19], Edwards [115] with a newer version Edwards2 released in 2018 and Stan [26]. This long list shows the interest and the potential of PP.

However, whereas on the software side research is currently moving forward at a fast pace, developments on the hardware side are still slow. Nonetheless, a few concepts have been presented in the literature. One approach is to perform Bayesian inference on analog signals, which can be seen as a hardware accelerator for belief propagation such as presented in [90]. Another example working on analog Bayesian inference is the work from Benjamin Vigoda presented in his thesis [117]. He proposed some analog components for his continuous-time circuits to perform various message-passing algorithms in binary factor graphs. His circuits were dedicated to wireless transceivers and were later developed by its company Lyrics, which has been acquired by Analog Devices. On the digital side, a Bayesian Computing Machine (BCM) has been proposed in [77]. It can be adapted to several applications, from AI to signal processing, and has been successfully tested on FPGA. Moreover, at circuit level, a probabilistic CMOS (Complementary Metal Oxide Semiconductor) has been proposed in [27]. They turned conventional CMOS into probabilistic CMOS by using noise. Furthermore, at transistor level, spintronic can push probabilistic hardware forward due to its intrinsic probabilistic properties. Therefore the p-bit (probabilistic bit) can be used as a building block for Bayesian inference hardware presented in [16].

The concept of using stochastic computing to perform probabilistic inference has been published by Eric Jonas in 2014 in his thesis [68]. To our knowledge, this concept has never been investigated before. The idea is to build dedicated hardware performing fast approximate Bayesian inference using components, which are intentionally stochastic. Several specific modules have been presented which produce samples from a conditional probability distribution based on stochastic computing. These gates can be scaled to face more challenging applications with several variables. Moreover, massive parallelism is used in the circuit to speed up the computation. For instance, a depth and motion perception task has been addressed. The corresponding probabilistic model has more than 10000 latent variables. Due to its simple low-power components, the system runs in real time with a 1000x speed up compared to conventional microprocessors. A second task has also been treated with a dedicated circuit. It consists of recognizing handwritten digits, from the MNIST database. The designed circuit achieved a 2000x speedup compared to the state of the art software solutions due to its parallelism.

The idea of using intentionally stochastic circuits to perform probabilistic inference is the concept we also use in our work.

2.7 Conclusion

In this chapter, we presented an overview of the related work on the hardware part of this thesis. First, we discussed the motivation of the development of new computing paradigms, which launched the rebooting community. Then, we reviewed the different approaches that has been proposed by the research community. Approximate computing uses conventional hardware but reduces power consumption by decreasing the accuracy of the data representation. Other approaches, such as quantum or optic computing are based on physical phenomena to accelerate computation. Moreover, by looking at the human brain, researchers developed neuromorphic computing hardware. Stochastic computing uses a different data representation than processors and aims to achieve more robust, low-power devices. Finally, we looked at probabilistic programming and the dedicated hardware, which has been proposed for probabilistic inference.

Chapter 3

Mathematical foundations

This chapter aims to define all the mathematical concepts needed to understand the upcoming chapters of this thesis. It is split into three parts. First, the principal concepts for probabilistic programming are explained. Second, the concept of exact inference is defined. Third, the idea of approximate inference is described.

There are by choice no examples in this chapter to help to understand the presented concepts since two detailed examples will illustrate the machines explained in the upcoming chapter. These examples are using the formalism defined in this chapter.

The first section in this chapter is mainly based on the book *Bayesian Programming* [18]. For more details, please refer to this book.

3.1 Bayesian programming

Mathematic models suffer from two main problems, which are incompleteness and uncertainty of the obtained sensor data. Therefore, probability theory is a mathematical structure, which unifies rational reasoning with the presence of both incompleteness and uncertainty [66].

In this chapter, we will introduce the key notions of Bayesian programming.

3.1.1 Discrete variables

In this work, we focus on problems with discrete and finite variables. There are three main categories of variables:

- S: the searched variables of which we are searching the probability distribution.
- K: the known variables which incorporate the known part of our model. They often consist of the sensor data.
- F: the free variables which are variables present in the model but are not related to the inference we are computing.

3.1.2 Probability

Probability quantifies the level of uncertainty given to the value of a variable. A probability is a value between 0 and 1. For example, the fact that the probability of a variable S will have the value s is 0.5 will be written as follows:

$$P(S = s) = 0.5$$

3.1.3 Normalization

The sum of the probabilities of all possible realizations of a variable is equal to 1. Therefore, for discrete variables one can write:

$$\sum_{s \in S} P(S = s) = 1 \tag{3.1}$$

However, we will us a more simplified notation:

$$\sum_{s} P(s) = 1 \tag{3.2}$$

3.1.4 Conditional probability

In probabilistic models you can have dependences based on conditions. A probability of a given variable may change depending on a value of another variable. Therefore we have the conditional probability: $P(S|K)$. This can be read as "probability of S conditioned on K". The sign | divides the variables into two parts. On the right are the variables with a defined value. On the left are the searched variables.

Based on the normalization rule, for every possible value k of K, we can write:

$$\sum_{s} P(s|K = k) = 1 \tag{3.3}$$

which can be abbreviated into:

$$\sum_{s} P(s|k) = 1 \tag{3.4}$$

3.1.5 Variable conjunction

Probabilistic variables can also be united in a so called conjunction. The conjunction of all the variables of a probabilistic model can be interesting to study. The conjunction is noted as the logic "and": $S \wedge K \wedge F$. When analyzing the conjunction of all variables, the possible values of the probability distribution of a conjunction is $Card(S) \times Card(K) \times Card(F)$ with $Card(S)$ the cardinality of the variable S. Note that the following notations are equivalent:

$$P(S \wedge K \wedge F) = P(SKF) = P(S, K, F) \tag{3.5}$$

For the sake of simplicity we will mainly use the latter one in this document. For every suitable triple (s, k, f), we will often use the shortening $P(S = s, K = k, F = f) = P(s, k, f)$

3.1.6 Bayes theorem

The Bayes theorem is a central theorem for Bayesian inference. It is based on the conjunction and is known in two forms. The first one:

$$P(S, K) = P(S) \cdot P(K|S) \tag{3.6}$$

$$= P(K) \cdot P(S|K) \tag{3.7}$$

The more famous form under which the Bayes theorem is known is:

$$P(S|K) = \frac{P(S) \cdot P(K|S)}{P(K)} \tag{3.8}$$

3.1.7 Marginalization rule

The marginalization rule is often used. It is based on the normalization and the Bayes theorem. It expresses:

$$\sum_s P(s, K) = P(K) \tag{3.9}$$

Using the normalization or the Bayes theorem it can be rewritten as:

$$\sum_s P(s, K) = \sum_s P(K)P(s|K) \tag{3.10}$$

$$= P(K) \sum_s P(s|K) \tag{3.11}$$

$$= P(K) \tag{3.12}$$

3.1.8 Joint distribution

The joint distribution of two probabilistic variables S and K is the distribution of their conjunction. The joint distribution fully determines the marginals, using the marginalization rule in section 3.1.7.

3.1.9 Decomposition

Defining the best way to calculate the joint distribution on a probabilistic model is the main challenge as one may want to have an easy way of computing the distribution using a good model and being able to learn it easily. Therefore, the decomposition is defined which rewrites the joint distribution as a product of simpler distributions.

3.1.10 Parametric form

Finally, to compute the inference on the decomposition, one has to define the mathematical form of each distribution present in the decomposition. In practice this step specifies the different distributions of the decomposition. Priors are often set to be uniform when one does not have any *a priori* knowledge on them.

3.1.11 Inference

After having defined the probabilistic model, its decomposition and the parametric form one has everything needed to calculate inference on that model. Computing an inference is similar to asking a question to the model.

Inference is based on logical deduction. Therefore, from the assumptions considered true one draws conclusions that are also considered true. In fact, computers are based on the same deducible logic, better known as Boolean algebra. It is also a way of making decisions based on binary true-false proposals.

Probabilistic inference can be seen as an extension of Boolean algebra since it allows uncertainty on variables to be considered and its premises to be updated. The hypothesis can still be considered as Boolean values. However, the conclusion can take all the values between zero and one. The term inference relates to drawing conclusions based on a series of hypotheses or observations. These conclusions take the form of probability distributions on the possible values of the inferred variables.

There are two types of inference in probability theory:

- Exact inference: it consists in using the properties described above to calculate the exact distribution.
- Approximate inference: it provides an approximation of the desired distribution. It is used when the cardinality of the searched variables is too large.

The next sections will present these two types of inference more in detail. Especially, the mathematical concepts for approximate inference are more deeply described.

3.2 Exact inference

Now that all the required tools for Bayesian programming have been defined, one want to compute the inference. Given the joint distribution

$$P(S, F, K) \tag{3.13}$$

One wants to compute:

$$P(S|K) \tag{3.14}$$

In practice, one always executes the same steps. First, using the marginalization rule, one can write:

$$P(S|K) = \sum_f P(S, f|K) \tag{3.15}$$

Second, using the Bayes theorem one obtains:

$$P(S|K) = \sum_f \frac{P(S, f, K)}{P(K)} \tag{3.16}$$

Finally, one replace the denominator by a constant independent of S:

$$P(S|K) = \frac{1}{Z} \sum_f P(S, f, K) \tag{3.17}$$

Using the decomposition, one can simplify the joint distribution $P(S, F, K)$ into a product of distributions easier to compute. One general decomposition is:

$$P(S, F, K) = P(S) \cdot P(K|S) \cdot P(F|K, S) \tag{3.18}$$

When computing the distribution $P(S|K)$, one has to compute the probability for each possible value of the searched variable S. The feasibility of this computation mainly depends on the number of values of S. In other words, it is possible to compute $P(S|K)$ for S having a small cardinality $Card(S)$. Computing $P(S|K)$ for each possible value is called exact inference. However, many problems have a large search space and in these cases exact inference becomes impractical. In case of too large search spaces, one may use approximate inference algorithms which are presented in the upcoming section.

3.3 Approximate inference: Sampling algorithms using Markov chains

When computing a probability distribution, the search space can be very large and hence it becomes impossible to compute the desired distribution using the exact inference described in the previous section. Therefore, a lot of researches have been done to develop methods for approximate inference. Among all these methods, we will focus on sampling.

This section is based on different books: [62], [20] and [72]. For more details, we refer to these books.

This section introduces some Markov chain Monte-Carlo (MCMC) methods. First, the Markov chains are described including the different properties of the transition matrix needed for further works. Second, the Perron-Frobenius theorem is presented ensuring stationarity of the Markov chain. Finally, some MCMC algorithms are presented such as the Metropolis algorithm, the Metropolis-Hastings algorithm, the Gibbs sampler.

3.3.1 Markov chains

Markov processes are random processes such that the next upcoming value of the process depends only on the current value and not on the previous ones. This property is called the **Markov property**.

Markov processes are used for stochastic simulation methods such as MCMC algorithms (Metropolis algorithm and Gibbs sampling), which will be presented in this document. Markov chains where first introduced in 1906 by Andreï Andreïevitch Markov. They are useful to study random processes due to their transition probabilities.

A Markov chain is specified by its state space. The state space defines the possible values that the process can take, each mapped to a state. To determine a Markov chain, we need a set of states $S = \{s_1, s_2, ..., s_e\}$. The given Markov process starts in a specific state s_i and then transits to a next state s_j given a certain probability p_{ij}. Due to the Markov property, the next state of the process only depends on the current state. Therefore, the probability to move from state s_i to state s_j is given by p_{ij}. Notice that it is possible that $i = j$ in order to stay in the same state. The probability p_{ii} gives the probability of this transition.

Transition matrix

The Markov transition matrix, also called Markov matrix or **stochastic matrix**, is a compact way of describing the transitions of a Markov chain. The matrix M is constituted of the transition probabilities of the Markov chain.

$$M = (p_{i,j}), \forall\, i,j \in [1,e] \quad p_{i,j} \geq 0$$

The entries of the matrix M are composed by the probabilities of moving from s_i to state s_j.

Using the transition matrix M, one can easily calculate the probability of being in one specific state s_j after n steps while being currently in state s_i which is an entry of the matrix M^n, namely $M^n_{i,j}$.

Similarly, calculating the probability of one state s_i after n steps using a given starting distribution stored in the vector v can be achieved using:

$$v^{(n)} = v.M^n$$

A stochastic matrix describing a Markov chain is a **right stochastic matrix**. Since the sum of the transition probabilities from a state i to all other states must be 1, this matrix is a right stochastic matrix, so that the elements of each row sum up to 1:

$$\sum_{j=1}^{e} M_{i,j} = 1$$

Moreover, any product of stochastic matrices is also a right stochastic matrix.

Matrix properties

Stochastic matrices can have several properties. The ones needed for our further work will be defined in the following.

M is **regular** (also often called **primitive**) if there exist a $k \in \mathbb{N}$ such that $M^k_{ij} > 0$ for every (i,j). A Markov chain is called a **regular chain** if some power of the transition matrix has only positive elements. Any transition matrix that has no zeros determines a **regular Markov chain**. However, it is possible for a regular Markov chain to have a transition matrix that has zeros.

M is **ergodic** (also called irreducible in some litterature) if for all (i, j) there exist a $k \in \mathbb{N}$ with $(M^k)_{ij} > 0$. A Markov chain is called an **ergodic chain** if it is possible to go from every state to every state (not necessarily in one move).

Notice that a regular matrix is ergodic. However the opposite statement is false like shown by this as example:

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.19}$$

It is clear that it is possible to move from any state to any state, so the chain is ergodic. However, if n is odd, then it is not possible to move from state 0 to state 0 in n steps. If n is even, it is not possible to move from state 0 to state 1 in n steps, so the chain is not regular.

For a matrix M, the set of eigenvalues is called the **spectrum** $\sigma(M)$ of M. The **spectral radius** ρ of M is defined as $\rho = \max\{|\lambda_1|, ..., |\lambda_n|\}$. An eigenvalue λ of M is **simple** if $dim_C \; ker(M - \lambda I_n) = 1$. The eigenvalue λ is **dominant** if $\forall \alpha \in \sigma(M) \setminus \{\lambda\}, |\lambda| > |\alpha|$.

3.3.2 Perron-Frobenius theorem

The Perron–Frobenius theorem, proved by Oskar Perron (1907) and Georg Frobenius (1912), asserts that a real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector can be chosen to have strictly positive components.

Let M be the stochastic matrix of the given Markov chain and $\rho = \rho(M)$ be spectral radius of M. The Perron-Frobenius theorem states that:

- Suppose M is regular, then $\rho > 0$, ρ is a dominant and simple eigenvalue of M. Furthermore, there exists a unique vector v, which is strictly positive such that $vM = \rho v$ and $||v||_1 = 1$
- Suppose M is ergodic, then $\rho > 0$ and ρ is a simple eigenvalue of M. Furthermore, there exists a unique vector v, which is strictly positive such that $vM = \rho v$ and $||v||_1 = 1$

Note that the matrix M from equation 3.19 is not regular but ergodic. Therefore ρ is not dominant.

The vector v is called the **Perron-Frobenius vector** of M.

3.3.3 Stationarity

Processes described by Markov chains can be stationary. The **stationary distribution** vector P is defined as a row vector that does not vary when applying the transition matrix M to it.

$$P \cdot M = P$$

The distribution vector P (also called the equilibrium distribution) is a row eigenvector of the transition matrix associated with eigenvalue 1.

Intuitively, a stochastic matrix represents a Markov chain. The application of the stochastic matrix to a probability distribution redistributes the probability mass of the original distribution while preserving its total mass. If the Markov chain is aperiodic and ergodic, it has a unique stationary distribution. And, when applying this process repeatedly, the distribution for the Markov chain converges to its stationary distribution.

The Perron-Frobenius theorem ensures that every ergodic stochastic matrix has such a stationary vector, and that the largest absolute value of an eigenvalue is always 1.

More formally, let P be the stationary distribution with $P = PM$. P is an eigenvector of M^k. Since M is a stochastic matrix, 1 is an eigenvalue of M. This means 1 is also the eigenvalue of M^k. As $||M^k|| = 1$, the spectral radius $\rho(M^k) = 1$. This means

$$1 \cdot (1 \ldots 1) = (1 \ldots 1) \cdot M^T$$

Due to the Perron-Frobenius theorem, if the transition matrix M is ergodic, there exists a unique stationary distribution P with $P_i > 0$ for all i. Furthermore, if M is regular, for every initial stationary distribution D, DM^k converges exponentially fast to P, in the sense that $||DM^k - P||_1 = O(\tau^k)$ with $\tau < 1$ defined as $\tau = max \{|\lambda| : \lambda \in \sigma(M), \lambda \neq 1\}$

While running a sampling algorithm to find the stationary distribution of a Markov chain, one can test the stationarity of the computed chain using the following algorithm. The goal is to find a distribution P' such that

$$|P'_i - P_i| \approx 0$$

To sample P', one uses this following algorithm:

1. Set the number of samples one want to make $N_{samples}$ and initialize $n = 1$
2. Choose a start state s_i. Set $N_i = 1$

3. Choose a state s_j which can be reached from s_i according to the transition probabilities stored in the line i of the transition matrix

4. Increase $N_j = N_j + 1$ and $n = n + 1$

5. Go back to step 3 until $n = N_{samples}$

6. Finally calculate $P'_i = \frac{N_i}{N_{samples}}$

3.3.4 Detailed balance

A Markov chain is called **reversible** if its stationary distribution P fulfills the detailed balanced equation:

$$\forall a, b : \; P(a) \; M_{ab} = P(b) \; M_{ba}$$

If a distribution P fulfills the detail balanced equation then it is a stationary distribution of M. However, while all ergodic Markov chains have a stationary distribution, all of them are not reversible.

Detailed balance implies that, around any closed cycle of states, there is no net flow of probability. For example, it implies that, for all states i, j and k,

$$M_{ij} M_{jk} M_{ki} = M_{ik} M_{kj} M_{ji}$$

Furthermore, for positive transition matrices, the condition of "net flow" implies detailed balance. Moreover, transition matrices that are symmetric always have detailed balance.

3.3.5 MCMC algorithms

Markov chain Monte Carlo (MCMC) methods are a class of algorithms for sampling from a probability distribution P based on constructing a Markov chain that has P as its stationary distribution. The state of the chain after a number of steps is then used to sample the desired distribution. The quality of the approximation of P improves as a function of the number of steps.

In this section we will look at three MCMC algorithms. First, the Metropolis algorithm. Second, the more general version of the previous algorithm, the Metropolis-Hastings algorithm. Third, a derived version of the Metropolis-Hastings algorithm, the Gibbs sampler.

3.3.6 Metropolis algorithm

We now introduce the Metropolis algorithm. The idea is to define a transition matrix M such that P is the stationary distribution of M, that is, such that $P = PM$. Let be X the state space of the Markov chain.

During each iteration of the Metropolis algorithm the following steps are computed:

1. Start with an initial element x and its probability $P(x)$.
2. Draw a new element x' from a uniform distribution on X. Its probability is $P(x')$.
3. If $P(x') > P(x)$, then output x'.
4. Otherwise, draw a random of $\alpha \in [0, 1]$.
5. Furthermore, compute $\beta = \frac{P(x')}{P(x)}$.
6. If $\beta < \alpha$, choose x' as a new sample.
7. Else, choose x.

When looking at the transition probability going from x to x', one sees that its value is:

$$M_{x,x'} = \frac{1}{\text{Card}(X)} \cdot \min\left[1, \frac{P(x')}{P(x)}\right] \quad if \frac{P(x')}{P(x)} \geq 1$$

Let us show that P is the stationary distribution of M. A sufficient condition for P to be the stationary distribution of the Markov chain with transition matrix M is that P verifies the detailed balanced equation. Let us consider two distinct states x and x' of the Markov chain having respectively $P(x)$ and $P(x')$ as probability values. Let us assume without loss of generality $P(x') \geq P(x)$. A simple computation shows that

$$P(x)M_{x,x'} = \frac{P(x)}{\text{Card}(X)} = P(x')M_{x',x}$$

and so P verifies the detailed balance equation:

$$P(x)M_{x,x'} = P(x')M_{x',x} \tag{3.20}$$

which also proves that P is the stationary distribution of the Markov chain defined by M.

3.3.7 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is a more general version of the Metropolis algorithm. In addition to the Metropolis algorithm, an *a priori* probability A is used modeling the probability of considering a specific transition.

Let $A(x'|x)$ be the probability of considering the transition of moving from state x to x'.

The *a priori* probability distribution A is used to select x' and in the step 5 of the algorithm when calculating β. Remember in the Metropolis algorithm β was calculated as follows:

$$\beta = \frac{P(x')}{P(x)}$$

In the more general case, the Metropolis-Hastings algorithm, β is computed using the *a priori* distribution:

$$\beta = \frac{P(x')}{A(x'|x)} \cdot \frac{A(x|x')}{P(x)}$$

Using the Metropolis-Hastings algorithm, the probability of going from x to x' is:

$$M_{x,x'} = A(x'|x) \cdot \min\left[1, \frac{P(x')}{A(x'|x)} \cdot \frac{A(x|x')}{P(x)}\right]$$

Again we can show that P verifies the detailed balanced equation of the Markov chain defined by M. Let us consider two states x and x' of the Markov chain having $P(x)$ and $P(x')$ as probability values. Let us assume without loss of generality:

$$\frac{P(x')}{A(x'|x)} \geq \frac{A(x|x')}{P(x)}$$

We have:

$$\begin{aligned} M_{x,x'} &= A(x'|x) \\ P(x)M_{x,x'} &= P(x)A(x'|x) \\ M_{x',x} &= A(x|x') \cdot \frac{P(x)}{A(x|x')} \cdot \frac{A(x'|x)}{P(x')} \\ P(x')M_{x',x} &= P(x)A(x'|x) \end{aligned} \tag{3.21}$$

and so P verifies the detailed balanced equation:

$$P(x)M_{x,x'} = P(x')M_{x',x} \tag{3.22}$$

a fact which also proves that P is the stationary of the Markov chain defined by M.

The difference between these two algorithms is the *a priori* distribution A. When looking at the factor β of the Metropolis-Hastings algorithm:

$$\beta = \frac{P(x')}{A(x'|x)} \cdot \frac{A(x|x')}{P(x)}$$

One notices that the *a priori* distribution A was added to the term. If A is symmetric : $A(x'|x) = A(x|x')$, then we obtain a generalization of the previous algorithm (the basic Metropolis algorithm) replacing the initial uniform choice of x' by A.

3.3.8 Connection between transition matrix and Markov chain

Looking back at the Metropolis algorithm, we will present an example to analyze the transition matrix of the built Markov chain.

Let us look at an example with a matrix of dimension 3. First, M is empty.

$$M = \begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \end{array}$$

While running the algorithm, let us look at the row M_{2j}. Let us assume for example $P(1) \geq P(2) \geq P(3)$. By the definitions above, $M_{21} = 1/3$ since $P(1) \geq P(2)$. Likewise, $P(3) \leq P(2)$, hence $M_{23} = 1/3 \cdot \frac{P(3)}{P(2)}$. Finally, $M_{22} = 1 - M_{21} - M_{23}$, hence $M_{22} = \frac{1}{3} + \frac{1}{3}(1 - \frac{P(3)}{P(2)}) = \frac{2}{3} - \frac{P(3)}{P3(2)}$. Now, the transition matrix looks like:

$$M = \begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} \cdots & \cdots & \cdots \\ \frac{1}{3} & \frac{2}{3} - \frac{P(3)}{P3(2)} & \frac{P(3)}{3P(2)} \\ \cdots & \cdots & \cdots \end{bmatrix} \end{array}$$

One sees that, indeed, the Metropolis algorithm uses a stochastic matrix.

3.3.9 Gibbs sampler

The Gibbs sampler is a special case of the Metropolis-Hastings algorithm. It has been introduced in 1984 in [54]. We will first look at the Gibbs algorithm itself before analyzing the similarity to the Metropolis-Hastings algorithm.

The basic idea is to split the complex search spaces into separate blocks to simplify the computation. We sample each block independently, conditioned on the most recent values of the other blocks. The Gibbs sampler simplifies a complex high-dimensional problem by breaking it down into more simple, low-dimensional problems.

Algorithm

The computation is done as follows:

1. Start with an initial sample $(x_1^0, ..., x_n^0)$ and $t = 0$
2. Choose i such that $i = t \bmod n$
3. Update all variables by $x_j^t = x_j^{t-1}$ except x_i^t which will be computed in the next step
4. Draw x_i^t according to $P(X_i | x_1^t, ..., x_{i-1}^t, x_{i+1}^t, ..., x_n^t)$
5. Set $t = t + 1$
6. Go back to step 2

One can check that the detailed balance equation holds for the Gibbs sampler.

3.4 Notations

We now introduce the notations we will use throughout this document. The notations are also reproduced in the appendix A. When using a capital letter, we denote the probabilistic variable: S. Small letter are used to express the value of a variable such as s for example. However, in many applications, we do not only have one variable and we can designate a variable as a conjunction of variables. In this case, we use bold capital letters: $\boldsymbol{S}$. Similarly to the one dimensional variable, we have a realization of the variable $\boldsymbol{S}$, which is expressed with $\boldsymbol{s}$. Note that $\boldsymbol{S}$ does often represent a full collection, often ordered as a vector or a matrix.

Chapter 4

Stochastic sampling machines for Bayesian inference

This chapter aims to introduce the stochastic machines for Bayesian inference. They will be used for the applications treated in this document. The mathematical concepts presented in the previous chapter are the basis of the machines. There are two types of machines, which are both using stochastic computing. The first one is called *Bayesian machine* and is dedicated to simple inference problems. The second one, called the *Bayesian sampling machine*, uses sampling techniques to solve high dimensional inferences.

Since both machines are based on stochastic computing, we will first introduce the concept of stochastic computing before presenting in detail the machines.

4.1 Stochastic computing

Both our Bayesian machines are based on stochastic computing. Probability values are encoded by streams of stochastic bits, drawn from a Bernoulli distribution. This concept has first been introduced in [51]. Each stochastic bit stream encodes a probability p, a number between 0 and 1[1]. Discrete temporal integration over n_T steps gives an approximation of p: this is done by counting the number n_1 of 1 and dividing by n_T, so we have:

$$\frac{n_1}{n_T} \xrightarrow[n_T \to \infty]{} p. \tag{4.1}$$

[1]Note that this is called unipolar stochastic computing in the literature [9]

A stochastic bit stream is a Bernoulli process which is based on a Bernoulli distribution with probability value p. The expectation, variance and standard deviation of a Bernoulli process are given by:

$$\mathrm{E}\left(\frac{n_1}{n_T}\right) = p \tag{4.2}$$

$$\sigma\left(\frac{n_1}{n_T}\right) = \sqrt{\mathrm{V}\left(\frac{n_1}{n_T}\right)} = \sqrt{\frac{p(1-p)}{n_T}} = \frac{\sqrt{p(1-p)}}{\sqrt{n_T}} \tag{4.3}$$

The standard deviation depends on the probability p. However, it highly depends on the length of the bit stream which is in $\mathcal{O}\left(\frac{1}{\sqrt{n_T}}\right)$. Moreover, $\mathcal{O}\left(\frac{1}{\sqrt{n_T}}\right)$ means a slow convergence due to the representation of the bit stream.

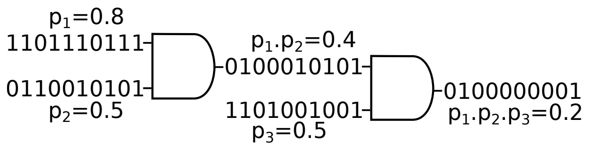


Figure 4.1: The multiplication of 3 probabilities p_1, p_2 and p_3 using stochastic bit streams.

The main advantage of the bit stream representation is that it allows to perform a probability product computation with a simple AND gate. Note that in order to multiply two stochastic bit streams, they need to be independent, which requires to generate them using two independent random number generators. As illustrated in figure 4.1, let p_1 and p_2 be two probability values respectively encoded by their bit stream chains B_1 and B_2. The result of their multiplication $p_1 \times p_2$ is the output of the AND gate, which takes B_1 and B_2 as inputs.

Stochastic computing is often used in approximate computing hardware since it allows to stop the computation at any time and adapt the precision of the result depending of the importance of the computation. In theory, stochastic bit streams have infinite lengths. In practice, the bit streams have a finite length.

A stochastic bit stream represents a probability distribution for value p, which varies with the length of the stream. This is illustrated by figure 4.2. We plotted the probability as a function of the given value. Note that the distributions are max-normalized. In this case we want to generate a stochastic bit stream $p = 0.8$. Therefore, we provide the probability for different bit stream lengths. In blue, a bit stream of length 5 has been

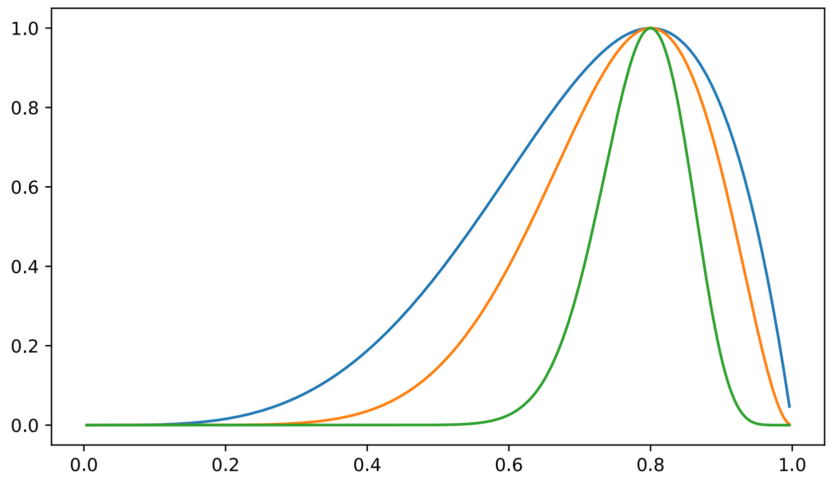


Figure 4.2: Probability of a value depending on the length of a stochastic bit stream for a bit stream representing the value 0.8. In blue, a 5 bits bit stream. In orange, a 10 bits bit stream. In green, a 40 bits bit stream.

generated. More precisely, in this stream, there are one "0" and four "1"s. One see that the distribution of is very broad and wide. The probability that others values can be deduced based on this bit stream is high. Moreover, in orange the distribution of a bit stream of a 10 bit length is shown. The bit stream is made out of 2 "0"s and 8 "1"s. One can see that the distribution is getting thinner. Finally, in green the distribution of a 40 bits bit stream is provided. The distribution gets peaky and hence the precision of the stochastic computing is more precise. Due to this figure, one can see how the precision of a bit stream evolves depending on its length. As a numerical representation bit streams are radically different from "classical" number representations. They are much more limited by their very nature. For instance, there is no equality operator for bit streams. Showing that this representation can be technically competitive is one challenge for our work.

In order to generate stochastic bit streams it is crucial to have a good enough random number generator (RNG). Otherwise, the computation using the stochastic bit streams is not valid. There are two main classes of random number generators (RNG), which are the *pseudo RNGs* (PRNG) and the *true RNGs* (TRNG). They mainly differ on the manner the random bits are generated and also on the quality of the generated numbers. Pseudo RNGs are based on algorithms and can hence be predicted as they are deterministic. On the other hand, true RNGs are based on physical phenomena and can not be predicted.

Several algorithms exist to generate random numbers. The random numbers generated by PRNGs typically have a period after which the generated numbers become redundant. Linear Feedback Shift Registers (LFSRs) are one of them.

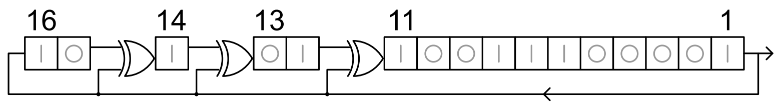


Figure 4.3: An example of a 16-bit Galois LFSR.

LFSR are linear feedback shift registers which means that the input bit of the shift register is a linear function of the previous state. We use the Galois LFSR which is using the exclusive-or (XOR) as linear function. Figure 4.3 shows an example of a 16-bit Galois LFSR [2]. In practice, the XOR of certain bits and of the overall output is used as input. The feedback polynomial specifies which bits are used for the XOR. In the example of the figure, the feedback polynomial is: $x^{16} + x^{14} + x^{13} + x^{11} + 1$. The value used to start the LFSR is called the seed. As the LFSR is deterministic, after a certain period, the output of the LFSR is repeated. This period depends on the feedback polynomial. For example, for the 16-bit Galois LFSR the maximum period one can achieve is a period of $2^{16} - 1$. The goal is to use LFSRs with a large enough period in order to avoid the repetition of the output. For our applications, we mainly use 32-bit Galois LFSRs.

4.2 Bayesian machine

In this section the first machine, the *Bayesian machine*, is introduced as well as all its components. It is dedicated to simple inference problems. In other words, it is usable for applications where the cardinality of the search variable S is small enough. This machine samples in parallel the distribution of S for all the possible values of S. As it is using stochastic computing, it estimates a kernel of the distribution. Therefore, when Card(S) becomes too big the size of the machine increases and becomes too large. Hence, the need of a sampling machine, which will be introduced in section 4.3, is mandatory.

[2]Original figure taken from `commons.wikimedia.org/w/index.php?curid=59986939`.

4.2.1 Bayesian fusion

In order to properly explain the Bayesian machine, let first define a model on which the machine could infer. Let us consider a discrete searched variable S, a discrete known variable K, and their joint distribution $P(S, K)$. S and K can be themselves conjunction of variables. Let us define the joint distribution:

$$P(S,K) = P(S,K^1,\ldots,K^n) = P(S)\prod_{i=1}^{n} P(K^i|K^1,\ldots,K^{i-1},S) \tag{4.4}$$

where $P(S)$ is the prior, $P(K^i|K^1,\ldots,K^{i-1},S)$ are the conditional distributions. The machine is dedicated to solve the following inference problems:

$$P(S|K^1,\ldots,K^n) = \frac{1}{Z}P(S)\prod_{i=1}^{n} P(K^i|K^1,\ldots,K^{i-1},S) \tag{4.5}$$

where Z is a normalization constant. One can see that the representation of the probabilistic model in such way leads to an inference, which can be made by multiplying all the terms.

In the case of naive Bayesian fusion, each conditional distribution is seen as a likelihood of one variable (so-called evidence) and (4.5) simplifies to:

$$P(S|K^1,\ldots,K^n) = \frac{1}{Z}P(S)\prod_{i=1}^{n} P(K^i|S). \tag{4.6}$$

The corresponding conditional discrete distribution $P(K^i|S)$ is defined according to the application. A typical case of Bayesian fusion is sensor data fusion.

4.2.2 Architecture of the machine

In this section, we present the architecture of the Bayesian machine (BM). It has originally been presented in [32].

First, for the sake of simplicity, let us focus on a specific value of the discrete search variable $S = s^j$. From (4.6), the machine approximates:

$$P(S=s^j|k^1,\ldots,k^n) = \frac{1}{Z}P(S=s^j)\prod_{i=1}^{n} P(k^i|S=s^j). \tag{4.7}$$

To be more precise, the BM estimates the kernel $\ker(s^j)$ of $P(S = s^j|k^1,\ldots,k^n)$. The kernel of a probability distribution is the form of the distribution in which any factors that do not depend on the probabilistic variables are neglected. They become part of the normalization factor of the probability distribution. For many applications, they are useless as the goal is often to localize the maximum of the distribution. In our case the kernel is:

$$Ker(s^j) = P(S = s^j) \prod_{i=1}^{n} P(k^i | S = s^j) \tag{4.8}$$

This computation (for line-index j) is represented on the second line of the left part of figure 4.4.

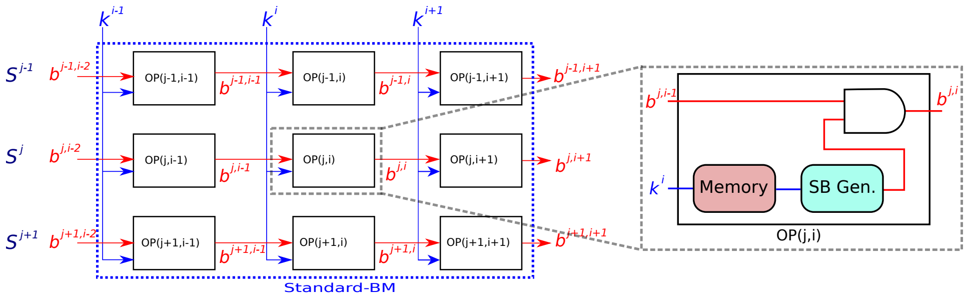


Figure 4.4: Architecture of the BM including a zoom on a single OP block of the machine. Bit streams are represented by red arrows. Blue arrows illustrate fixed-point numerical values.

Indeed, as shown in this figure, the architecture of the machine is shaped as a matrix of elementary blocks. The OP block will be described in detail in the upcoming section. Every block represents an individual probability product operator. Each evidence k^i is associated to a column. Hence, the different columns correspond to different evidences. In the case of a sensor fusion problem, each column gets the data of one sensor. The different lines of the matrix correspond to the different values of the search variable S. In summary, each column-wise process updates the current state of knowledge with a new evidence.

The machine generates samples at each calculation step, for each value of the searched variable. For each line, i.e for each value of the searched variable s^j, the result of the n cascaded AND gates after n_T calculation steps is accumulated in counters which are located at the end of each line, as shown in figure 4.5. This discrete temporal integration allows to recover the target kernel. An approximation of the kernel $P(S = s^j | k^1, \ldots, k^n)$ may be obtained by normalizing the counter values $\frac{counter_j}{\sum_l counter_l}$.

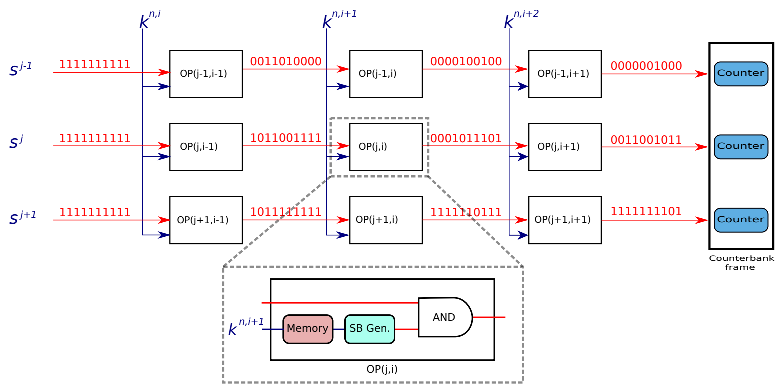


Figure 4.5: Detailed overview of the Bayesian machine architecture including the counters.

4.2.3 OP block

This section aims to describe in detail the components and the function of the OP block in the Bayesian machine.

The machine is organized as a matrix, as shown in figure 4.4. Typically, a block at position (j, i) in the matrix takes as inputs:

1. the evidence value k^i
2. the stochastic bit stream $b_{j,i-1}$ (in red in the figure) representing the product (4.7) up to index $i-1$, which is the output of the previous column for the same line.

The right part of figure 4.4 details the basic operator (OP). It is composed of the AND gate to perform the stochastic product between $b_{j,i-1}$ and the stochastic bit stream representing the likelihood value $P(K^i|S)$ corresponding to the evidence k^i. The likelihood is stored in the memory block of the OP block. This memory can be programmed depending on the application that is treated by the machine. Then, a Stochastic Bit stream Generator (SBG) generates samples according to a Bernoulli distribution corresponding to the likelihood value. The SBG is composed of a random number generator (RNG), which is in our case a 32-bit Galois LFSR. The output of the RNG is compared to the likelihood value to generate the correct Bernoulli distribution.

Figure 4.6 provides and example of the SBG which is using a 16-bit Galois LFSR. We do not take the output of the LFSR, which is normally only one bit. But instead we use the 8 first bits which are currently in the registers (in

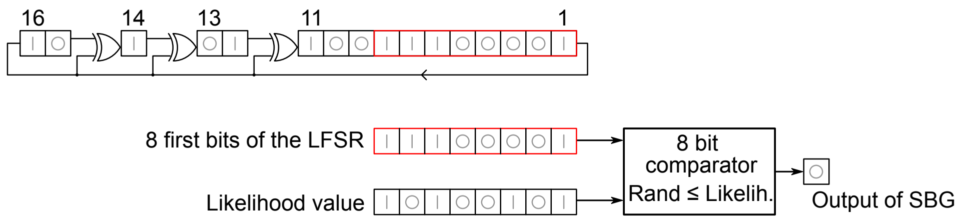


Figure 4.6: Example of an SBG using a 16-bit Galois LFSR.

red in the figure). These 8 bits are compared to the likelihood value, which is stored in a register. We choose to encode our likelihood values in 8 bits. To generate the output of the SBG, we use a comparator, which compares the 8 bits of both numbers and generates the output. The obtained stochastic bit is then used as input of the AND gate to perform the multiplication. The output of the AND gate is the new bit stream $b_{j,i}$, shown in figure 4.8.

However, since in all stochastic computing architectures, the computation using AND gates is valid if the AND gate is processed with two independent bit streams. Thus, one can not use the same Random Number Generator (RNG) for these two bit streams. Note that if this hypothesis is not fulfilled, the computation can heavily fail.

Using LFSRs to generate stochastic bit streams can lead to faulty bit streams. This concern is discussed in appendix B.

4.2.4 Tackling temporal dilution

The data representation used in our Bayesian machines is one of the strengths since it helps to substantially accelerate the computation. However in certain situations, this strength can quickly become a downside of the system. This problem is called the *temporal dilution*.

In this section, we first explain this phenomenon before describing the mechanism that we implemented to tackle the temporal dilution. The Bayesian machines often suffer from the data representation they use, namely the stochastic bit streams. All probabilities encoded in the machine are represented by a stochastic bit stream for the computation. The stochastic bit streams pass through a lot of AND gates in our machines. This inevitably leads to a decrease of the number of "1" in the output since the probability gets smaller after the multiplication. We call this effect the *temporal dilution*. In particular, since we mainly deal with low probability values, the bit stream representation requires long streams to represent such small values.

Figure 4.1 illustrates this problem. The product between $p_1 = 0.8$,

$p_2 = 0.5$ and $p_3 = 0.5$ is performed with two cascaded AND gates. The resulting bit stream, encoding $p_1 \times p_2 \times p_3 = 0.2$, is composed of only two "1" in a chain of 10 bits.

Moreover, when the number of AND gates in one line increases, this problem gets worse. Typically, our machines typically have 100 columns in our standard case (see chapter 6), which means that in each line of the machine, there are 100 AND gates to multiply the probabilities. Thus, the temporal dilution is present and rapidly leads to an increase of computation time of several thousands of steps.

4.2.5 Max-Normalization

Knowing that the temporal dilution has an heavy impact on the performance of the machine, we propose a mechanism to deal with that issue. The goal is to increase the number of "1"s in the stochastic bit streams. It is crucial to not modify the final probability distribution. However, we can modify the global normalization term $1/Z$ that can be found in equation (4.6).

Therefore, we propose to column-wise maximize each set of probability values (priors and likelihoods) while keeping unchanged the ratios between the different values. To this aim, each prior value is normalized by the maximum of all the prior values of the same column and the same is done for every column of the likelihoods:

$$\begin{aligned} &\text{Max-normalized priors} : \frac{P(S = s^j)}{\underset{s \in S}{Max}\{P(S = s)\}}, \\ &\text{Max-normalized likelihoods} : \frac{P(K^i = k^i | S = s^j)}{\underset{s \in S}{Max}\{P(K^i = k^i | S = s)\}}. \end{aligned} \tag{4.9}$$

This process is referred to as max-normalization. At the end of each line of the BM computation matrix, one can evaluate the max-normalized kernel of the distribution:

$$Ker(s^j) = \frac{P(S = s^j)}{\underset{s \in S}{Max}\{P(S = s)\}} \prod_{i=1}^{n} \frac{P(k^i | S = s^j)}{\underset{s \in S}{Max}\{P(k^i | S = s)\}}. \tag{4.10}$$

Therefore, the "true" posterior probabilities can be obtained by conventional normalization of (4.10). However, most of the time, the computation done by the machine, which is proportional to the desired distribution, is enough since one mainly wants to find the maximum of the distribution.

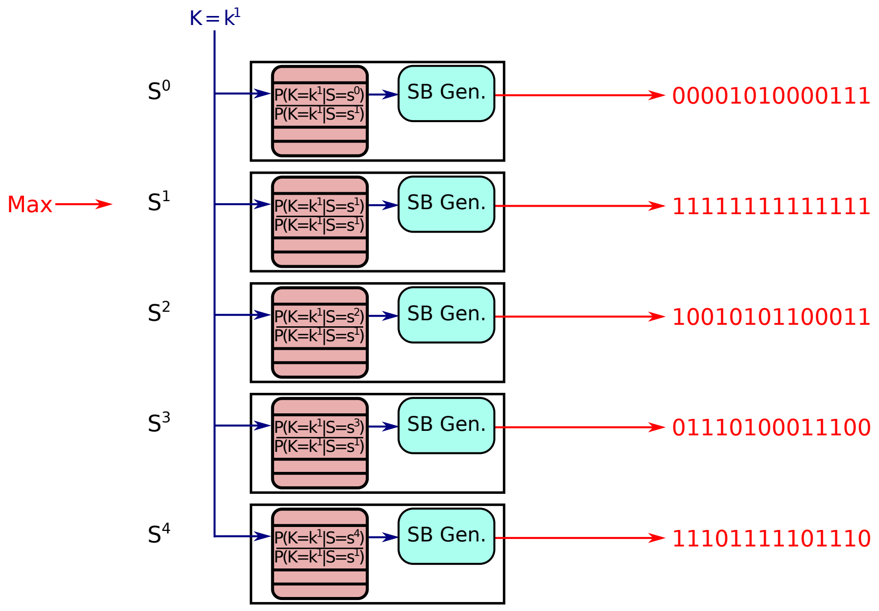


Figure 4.7: Example of a max-normalization on a column of 5 lines.

To demonstrate the efficiency of the max-normalization, let us take a simple example with a uniform distribution on priors and likelihoods. For simplicity, let w be the dimension of the search space and let n be the number of evidences corresponding to the number of columns of the matrix. Assume the probability of generating a "1" for each prior and each likelihood is $p = \frac{1}{m}$. Through the n AND gates, the probability for each line to finally emit a "1", and fill the corresponding counter, is $p_{out} = \frac{1}{m^{n+1}}$. This probability quickly tends towards "0" even with small values for n and m. In this case, the machine needs a very long time to obtain useful information incrementing the counters. Yet, in this example, the important information is that all counters encode the same value. Using the max-normalization technique over both priors and likelihoods, each maximum of each probability distribution are equal to 1. Then, all the corresponding OP blocks of the matrix output a bit stream only composed of "1" encoding the value 1.0. At each computation step, all counters are incremented, hence the values of the different counters remain equal. To get the approximate value of $P(S = s^j | k^1, \ldots, k^n)$, again, we compute the ratio $\frac{counter_j}{\sum_l counter_l} = \frac{counter_j}{m \times counter_j} = \frac{1}{m}$ which is the expected result of the uniform law.

To illustrate the max-normalization, figure 4.7 gives a small example of a column containing 5 lines. In the memory block, the normalized probability values are stored. Over the 5 lines, the line S^1 is the one which has the highest probability value $P(K = k^i|S = s^1)$. For that reason every line is divided by $P(K = k^i|S = s^1)$:

$$\frac{P(K = k^i|S = s^i)}{P(K = k^i|S = s^1)}$$

Therefore, as shown in the figure, the line S^1 provides an output bit stream containing only "1"s since it contains the maximum.

Moreover, note that in figure 4.5 the priors, which are sent to the column of the machine are all equal to 1. This is due to the fact that most of the time we do not have any prior knowledge on the search distribution and hence let it follow the uniform distribution. Once max-normalized, the uniform distribution remains in sending "1"s in all the stochastic bit streams in all lines.

4.2.6 Application: boat localization

In this section, we go through an example of an application which uses the Bayesian machine. The task is to localize a boat based on sensors which are capable of getting an angular and a distance information of the three lighthouses placed along the coast. This problem is a sensor fusion where one need to fuse the data of the six noisy sensors to estimate the boat position. The setup is shown in figure 4.8

The area where the boat is localized is discretized into a grid of $16 \times 16 = 256$ cells. The lighthouses are located at the origin and at the middle of the X and Y-axes.

We have to introduce the probabilistic model. First the needed probabilistic variables which are:

- X: the variable associated to the coordinate of the boat on the X-axis. This value is discretized on 16 different values.
- Y: the variable associated to the coordinate of the boat on the Y-axis. This value is discretized on 16 different values.
- B_1, B_2, B_3: the variables associated to the sensor values of the angular (bearing) sensor
- D_1, D_2, D_3: the variables associated to the sensor values of the distance sensor

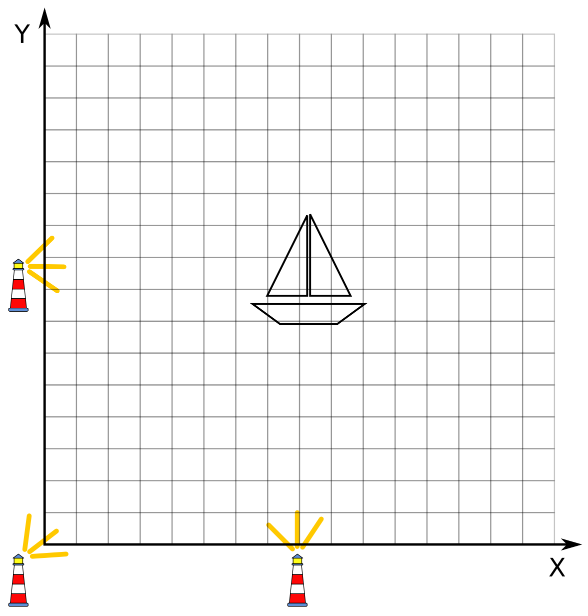


Figure 4.8: Setup for the boat localization example with a 16 by 16 grid.

The joint distribution of our model is:

$$P(X, Y, B_1, B_2, B_3, D_1, D_2, D_3) \tag{4.11}$$

The decomposition can be written as follows:

$$P(X, Y, B_1, B_2, B_3, D_1, D_2, D_3) = P(X, Y) \prod_{i=0}^{3} P(B_i|X, Y) \cdot P(D_i|X, Y) \tag{4.12}$$

As we do not have any *a priori* knowledge on the position of the boat, $P(X, Y)$ is set as a uniform distribution. The other distributions $P(D_i|X, Y)$ and $P(B_i|X, Y)$ are assumed to follow a normal distribution. In summary, the decomposition is based on the following distributions:

$$P(X, Y) = \mathcal{U} \tag{4.13}$$

$$P(D_i|X, Y) \sim \mathcal{N}(\sqrt{(X - X_o)^2 + (Y - Y_o)^2}, \sigma^2_{\text{dist}}) \tag{4.14}$$

$$P(B_i|X, Y) \sim \mathcal{N}(\arctan \frac{X - X_o}{Y - Y_o}, \sigma^2_{\text{bear}}) \tag{4.15}$$

where X_o and Y_o are the coordinates of the sensor o.

For the inference, as we want to estimate the position of the boat, we want to compute the distribution $P(X, Y|B_1, B_2, B_3, D_1, D_2, D_3)$. Therefore, using the Bayes theorem, the condition distribution can be written as:

$$P(X, Y|B_1, B_2, B_3, D_1, D_2, D_3) = \frac{1}{Z} P(X, Y) \prod_{i=0}^{3} P(B_i|X, Y) \cdot P(D_i|X, Y) \tag{4.16}$$

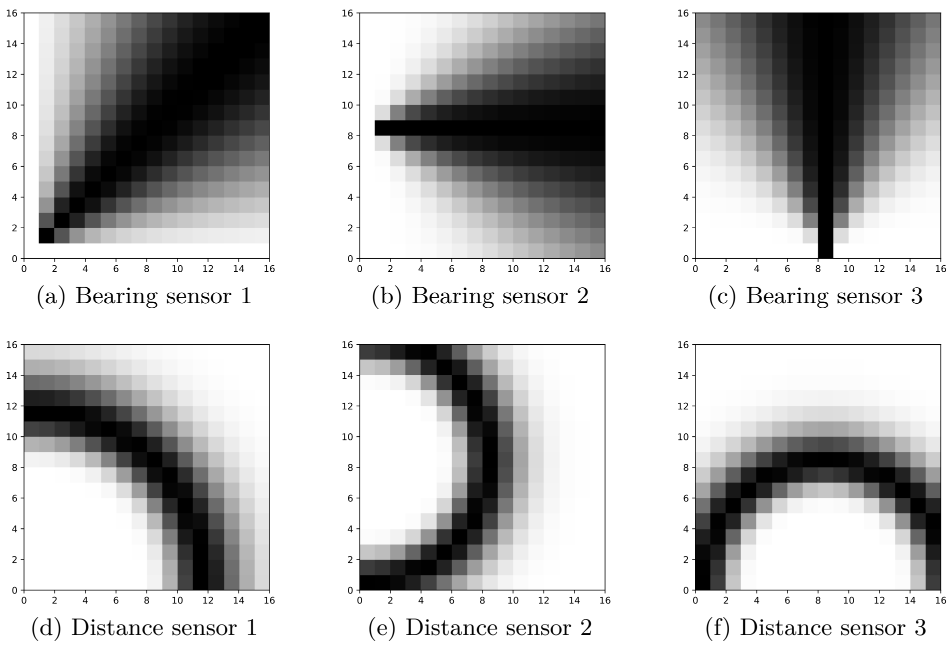
where Z is a normalization constant. For the inference we will focus on simplified form which is:

$$P(X, Y|B_1, B_2, B_3, D_1, D_2, D_3) \propto P(X, Y) \prod_{i=0}^{3} P(B_i|X, Y) \cdot P(D_i|X, Y) \tag{4.17}$$

In total there are six multiplications to perform to compute the probability for one position (x, y) given the sensor values $b_1, b_2, b_3, d_1, d_2, d_3$. As our grid has 16 by 16 cells, we have in total 256 positions to evaluate and compute the formula (4.17). Thinking about the Bayesian machine, it has 6 columns (one for each sensor) and 256 lines (one for each position of the grid). The likelihoods are computed according to (4.13) and max-normalized as explained in section 4.2.5.

To illustrate the task of the machine, figure 4.9 shows the max-normalized likelihoods of the different sensors based on the evidences (or recorded sensor values). The three bearing sensors (figures 4.9a, 4.9b and 4.9c) provide an angular information which becomes uncertain as the distance to the boat increases. Moreover, the three distance sensors (figures 4.9d, 4.9e and 4.9f) provide a distance information. The goal is to fuse all these sensor information and estimate the position of the boat.

As the Bayesian machine uses stochastic computing, it is possible to adapt the precision of the computed distribution by modifying the length of the stochastic bit streams. Therefore, one can plot the computed distribution for the different bit stream lengths. Figure 4.10 shows the estimated kernel by the Bayesian machine for different lengths of the stochastic bit stream. After a few steps, the maximum of the estimated kernel is located at the boat position as shown by the figures 4.10a and 4.10b. The more the machine runs, the more precise the estimated kernel is. Figures 4.10c, 4.10d and 4.10e show how the kernel fine-tunes and the final result takes shape. After 50 steps, the result slightly changes but the shape of the kernel is not modified as shown by the figures 4.10f to 4.10i. Note that after 1000 steps, the final result is obtained and the kernel does not change anymore. This example has been presented in [32] and in [18].


(a) Bearing sensor 1 (b) Bearing sensor 2 (c) Bearing sensor 3

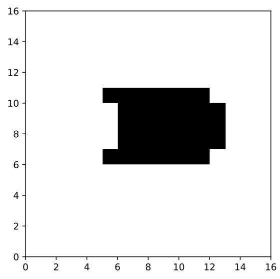
(d) Distance sensor 1 (e) Distance sensor 2 (f) Distance sensor 3

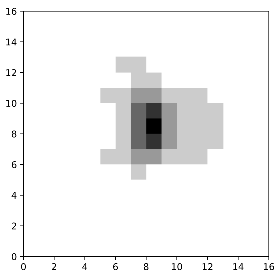
Figure 4.9: Distribution of the different sensors which are fused by the Bayesian machine.

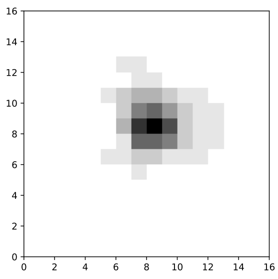
4.3 Approximate inference - Bayesian sampling machine

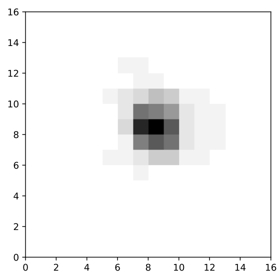
Until now the Bayesian machine has been presented which can be used for simple inference problems where the searched variable has a small cardinality. However, the number of lines in the Bayesian machine (which each represents a value of the searched variable) can not increase indefinitely. Hence, there is a need of another type of machine dedicated to high dimensional problems. In this section, we introduce the Bayesian sampling machine, which is based on a sampling method. This machine uses the Gibbs sampler, which has been introduced in chapter 3. The goal is to estimate a probability distribution when the dimension of the searched variable becomes too large for our Bayesian machine introduced in section 4.2.2. Therefore, sampling the distribution becomes the only possible way.

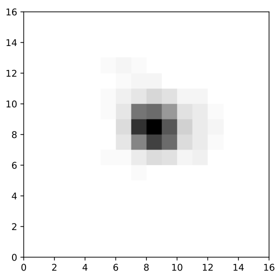
Like the Gibbs sampler, the machine focuses on one dimension of the searched variable at a time. Moreover, in presence of free variables F, the searched variable can be seen as a conjunction of the search S and the free

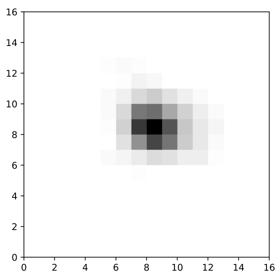

(a) Result after 1 bit

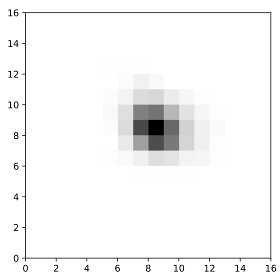

(b) Result after 5 bits

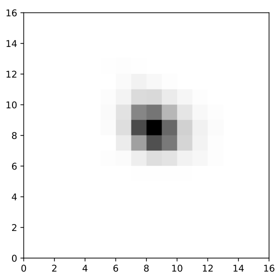

(c) Result after 10 bits


(d) Result after 20 bits


(e) Result after 50 bits


(f) Result after 100 bits


(g) Result after 200 bits


(h) Result after 500 bits

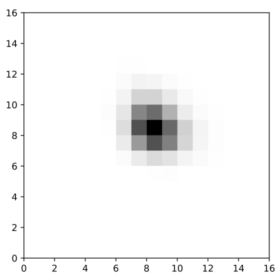

(i) Result after 1000 bits

Figure 4.10: Computed distribution for different lengths of bit streams.

F variables. Let be $P(\boldsymbol{S}|\boldsymbol{K})$ the distribution one wants to sample. The Gibbs algorithm will draw from the following distribution for each value of $u \in \{1, ..., \mathrm{Card}(\boldsymbol{S})\}$:

$$P(S^u|\boldsymbol{s}^{\neq u}, \boldsymbol{K}) \tag{4.18}$$

where $\boldsymbol{s}^{\neq u}$ is the set of variables including all variables $\boldsymbol{s}$ except the value $\boldsymbol{s}^u$. For example, the first four steps of the Gibbs algorithm will be drawing respectively from these distributions:

$$P(S^1|\boldsymbol{s}^{\neq 1}, \boldsymbol{K}) = P(S^1|s^2, s^3, s^4, \boldsymbol{K}) \quad (4.19)$$
$$P(S^2|\boldsymbol{s}^{\neq 2}, \boldsymbol{K}) = P(S^2|s^1, s^3, s^4, \boldsymbol{K}) \quad (4.20)$$
$$P(S^3|\boldsymbol{s}^{\neq 3}, \boldsymbol{K}) = P(S^3|s^1, s^2, s^4, \boldsymbol{K}) \quad (4.21)$$
$$P(S^4|\boldsymbol{s}^{\neq 4}, \boldsymbol{K}) = P(S^4|s^1, s^2, s^3, \boldsymbol{K}) \quad (4.22)$$
$$(4.23)$$

Once each of the searched variables has been drawn once (for each $u \in \{1, ..., \text{Card}(\boldsymbol{S})\}$), one so called sweep of the Gibbs algorithm is finished. In one sweep of the Gibbs algorithm, the machine will draw once from each distribution for every value of $u \in \{1, ..., \text{Card}(\boldsymbol{S})\}$.

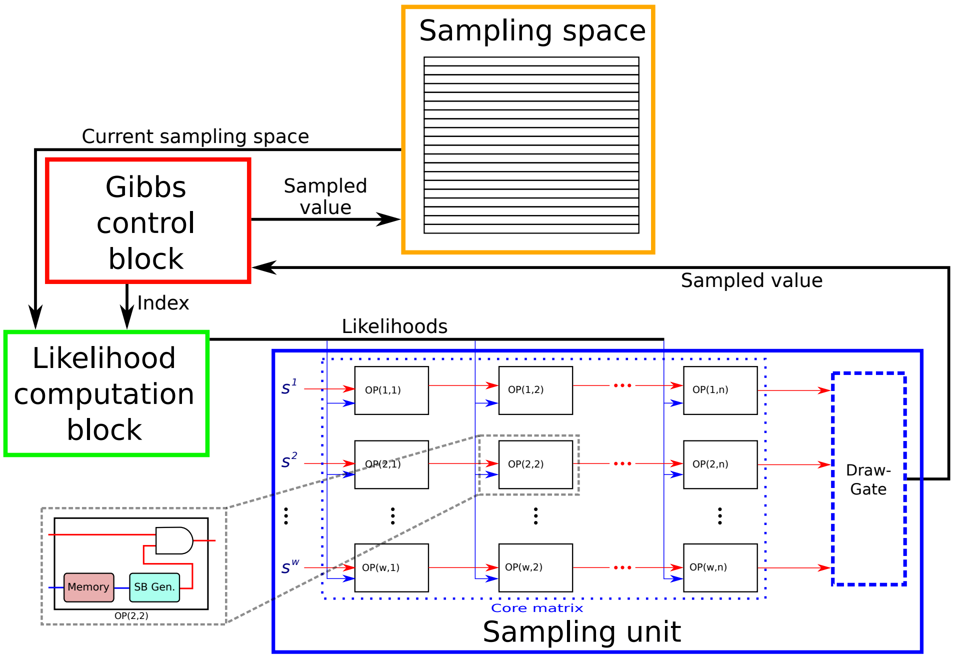


Figure 4.11: Architecture of the Bayesian sampling machine.

The Bayesian sampling machine implements the approach of the Gibbs sampling which is by drawing step by step of the distributions. Figure 4.11 shows the overall architecture of the machine. Its main component is the sampling unit (in blue) which is concerned with each distribution $P(S^u|\boldsymbol{s}^{\neq u}, \boldsymbol{K})$. The sampling unit is organized as the core of the Bayesian machine (BM) introduced in section 4.2. However, contrary to the BM, here the counters at the end of each line are replaced by the draw-gate. As we have to draw from the distribution and do not need to compute the entire distribution for all the values of $\boldsymbol{S}$, one sample of the distribution is sufficient. In other words, the draw-gate waits for one bit at "1" in a stochastic bit stream among all the lines of the core matrix of the sampling unit. Once at least one line has sent a "1" in its stochastic bit stream to the draw-gate, the draw-gate is activated and provides the desired sample s^u. The detailed way of operating will be explained in the upcoming section, section 4.3.1. This sample s^u is stored in a dedicated memory which stores all the current sampled values (sampling space in orange). Then, the Gibbs control block changes the index u and the likelihoods for the new distribution are computed to run the sampling unit to draw from the new distribution. Note that at each step, the machine takes all the current values in the sampling space into account. Hence, the need of an on-chip likelihood computation unit is required, which is not needed in case of the Bayesian machine.

4.3.1 Draw-gate

The draw-gate is one of the principal modules in the Bayesian sampling machine. The core of the sampling machine is stopping when only one line is set to "1". The corresponding value is chosen as the new sample s^u. However, the output of the draw-gate when several lines are activated simultaneously, needs to be defined.

Let X_t be the output of the draw-gate at time step t. Consider now the draw-gate to have w input lines, indexed by k in $\mathbb{Z}_w = \{1, 2, \ldots, w\}$. The lines correspond to the w possible values of the variable S^u. All lines of the core matrix produce a w-bits stream $(B_t)_{t\geq 0}$ at each step of the machine.

Each w-tuple of bits $B_t = B_t^1 B_t^2 \cdots B_t^w$ is in $\mathbb{B}_w = \{0,1\}^w \setminus \{(0, 0, \ldots, 0)\}$. Each time some B_t is sent to the draw-gate, the draw-gate emits one single signal X_t in $\mathbb{Z}_w$, defined as

$$X_t = \min\{k \geq X_{t-1} + 1 \mid B_t^k = 1\}$$

Here, an important convention is that one examines the values B_t^k starting immediately after the position X_{t-1}, in their order of succession for the natural succession order on the discrete circle $\mathbb{Z}_w$ (thus, each $k \neq w$ in $\mathbb{Z}_w$ is followed by $k+1$ and w is followed by 1), and that one stops at the position of the first 1 that one encounters (there is always at least one). This line index is set as the output X_t of the draw-gate.

The process $(X_t)_{t\geq 0}$ is such that $X_t = F_n(X_{t-1}, B_t)$, for some function $F_w : \mathbb{Z}_w \times \mathbb{B}_w \to \mathbb{Z}_w$ which can be explicitly written down.

Let us consider a little example with three lines ($w = 3$). In this case, the function F_3, which provides the output of the draw-gate is defined as shown in table 4.1.

B_t	$X_{t-1} = \mathbf{1}$	$X_{t-1} = \mathbf{2}$	$X_{t-1} = \mathbf{3}$
001	3	3	3
010	2	2	2
011	2	3	2
100	1	1	1
101	3	3	1
110	2	1	1
111	2	3	1

Table 4.1: Output of the draw gate in case of $w = 3$. Each line of the table is associated to a 3-bit input to the draw-gate at time t. The output is given depending on the previous output of the draw-gate X_{t-1} which is associated to the columns of the table.

The formal mathematical proof of that this algorithm indeed yields the desired distribution is provided in appendix C. It is due to Didier Piau, as a member of the MicroBayes project.

4.3.2 Application: boat localization with a large grid

In this section, an example is explained in order to illustrate the Bayesian sampling machine. The example is based on the boat localization problem which has been exposed in section 4.2.6. However, in this case, the grid becomes much larger and hence the need for the Bayesian sampling machines becomes real. In this example a grid of 1024×1024 is used and the six sensors are as in the previous setup of this problem in section 4.2.6 (the sensors 2 and 3 are located at the middle of the X-axis and Y-axis).

Figure 4.12 shows the chosen setup. Note that the grid is much bigger than on the picture. Moreover, the boat is located in the middle of the grid, at cell (512,512).

Again, the goal is to compute the follow distribution:

$$P(X, Y | B_1, B_2, B_3, D_1, D_2, D_3) \tag{4.24}$$

However, since the cardinality of $X \wedge Y$ is too large, we can not compute the distribution using exact inference methods. Therefore, for one sweep of

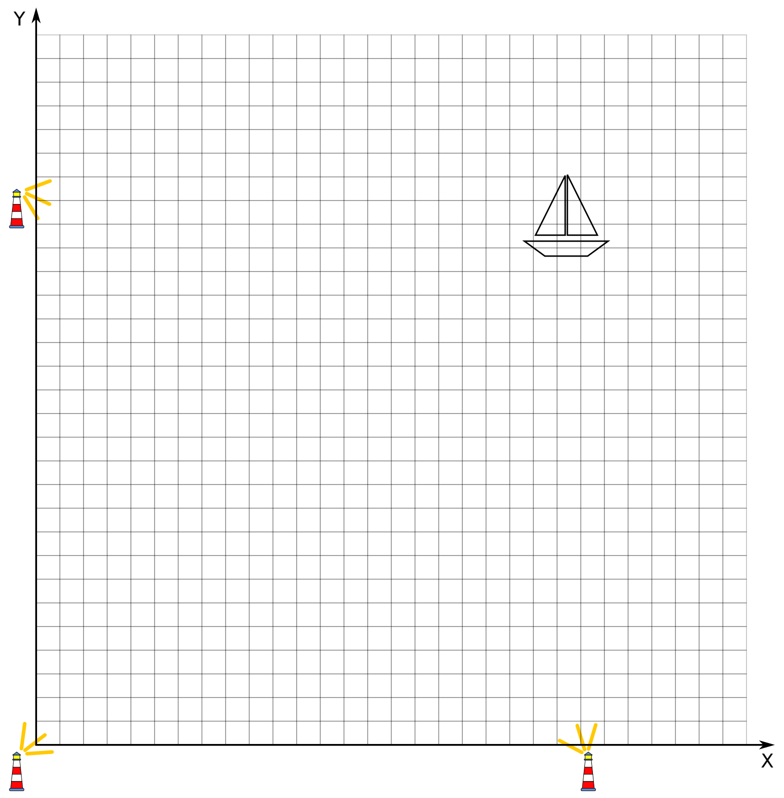


Figure 4.12: Setup for the boat localization example with a 1024 by 1024 grid.

the Gibbs algorithm, using the current value for y, we draw from:

$$P(X|y, b_1, b_2, b_3, d_1, d_2, d_3). \tag{4.25}$$

We use the sampling unit of the Bayesian sampling machine to perform this draw. Each column in the matrix of OP blocks in the sampling unit represents one sensor $(b_1, b_2, b_3, d_1, d_2, d_3)$. Each line corresponds, in this particular case, to each value of the searched variable X. Then, taking into account the drawn value for x in the previous step, we draw from:

$$P(Y|x, b_1, b_2, b_3, d_1, d_2, d_3) \tag{4.26}$$

The draw unit launches the computation of the kernel which results in 6 multiplications of the max-normalized likelihoods to fuse the data of the six sensors.

$$Ker(X|y, b_1, b_2, b_3, d_1, d_2, d_3) \propto P(X, y) \prod_{i=0}^{3} P(B_i|X, y) \cdot P(D_i|X, y) \quad (4.27)$$

The different distributions needed to compute the likelihoods are defined as follows:

$$P(X, Y) = \mathcal{U} \quad (4.28)$$
$$P(D_i|X, Y) \sim \mathcal{N}(\sqrt{(X - X_o)^2 + (Y - Y_o)^2}, \sigma^2_{\text{dist}}) \quad (4.29)$$
$$P(B_i|X, Y) \sim \mathcal{N}(\arctan \frac{X - X_o}{Y - Y_o}, \sigma^2_{\text{bear}}) \quad (4.30)$$

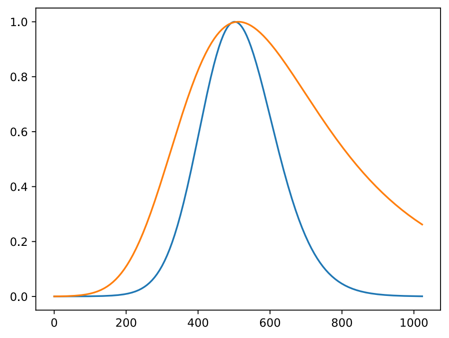
where X_o and Y_o are the coordinates of the sensor o and x or y is fixed.

This results in sampling the value of Y given a currently fixed value of x. Figure 4.13 shows the likelihoods for the different sensors in the situation where the current value of y is 512 and we are drawing $P(X|y = 512, b_1, b_2, b_3, d_1, d_2, d_3)$. In blue are the likelihood distributions $P(D_i|X, Y = 512)$ for the distance sensors whereas in orange are the likelihood distributions $P(B_i|X, Y = 512)$ of the bearing sensors. One can see that they are all Gaussian distributions except in figure 4.13c for the bearing sensor located in (512,0) which is uniform since we are currently sampling the column which is in front of the sensor.

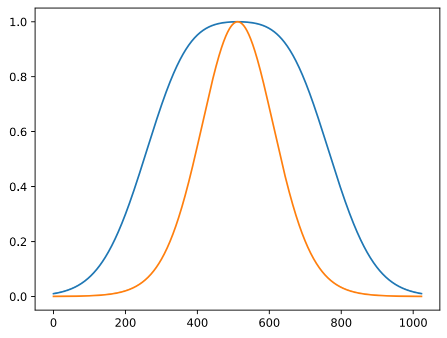
When running the system for several sweeps, one can plot the sampled positions which were drawn according to the Gibbs algorithm. Each step in the plot represents one sweep of the algorithm. Note that the initial position is always chosen randomly. The plot of the exploration is provided in figure 4.14. The system runs for 20 sweeps and quickly converges to the desired position in (512,512).

4.4 Conclusion & discussion

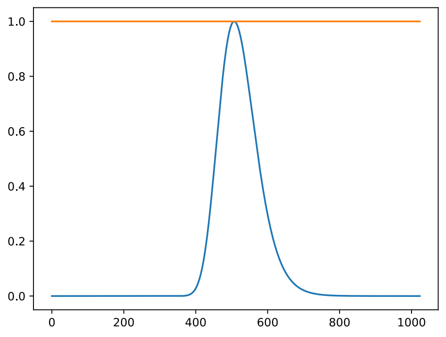
In this chapter, both our Bayesian machines have been described. They are both based on stochastic computing. The first one, called the *Bayesian machine*, is dedicated to simple inference problems, which are characterized by the searched variables having a small enough cardinality. The second one, called the *Bayesian sampling machine*, is more generic and is used to solve higher dimensional inferences. Examples have been presented to explain the different machines.



(a) Likelihood for sensor located in (0,0)



(b) Likelihoods for sensor located in (0,512)



(c) Likelihoods for sensor located in (512,0)

Figure 4.13: Plot of the likelihood distributions for the different sensors. In blue the distance sensors. In orange the bearing sensors.

Although stochastic computing has historically failed compared to other computing methods, it has several strengths beside it drawbacks. Since our machines are using stochastic computing, we aim to optimize our architectures to use the strengths of randomness while minimizing its weaknesses. More precisely, stochastic computing allows to perform a multiplication of two probabilities using a simple AND gate. Since for our inferences, we need to compute a lot of multiplications, we use stochastic computing. However, when representing a probability using a stochastic bit streams and using an AND gate to multiply two probabilities, the computing could require a lot of time, especially in case of low probabilities. Since in each column, two probabilities are multiplied and the result is a multiplication of all the likelihoods, the result decreases exponentially with the number of columns. In

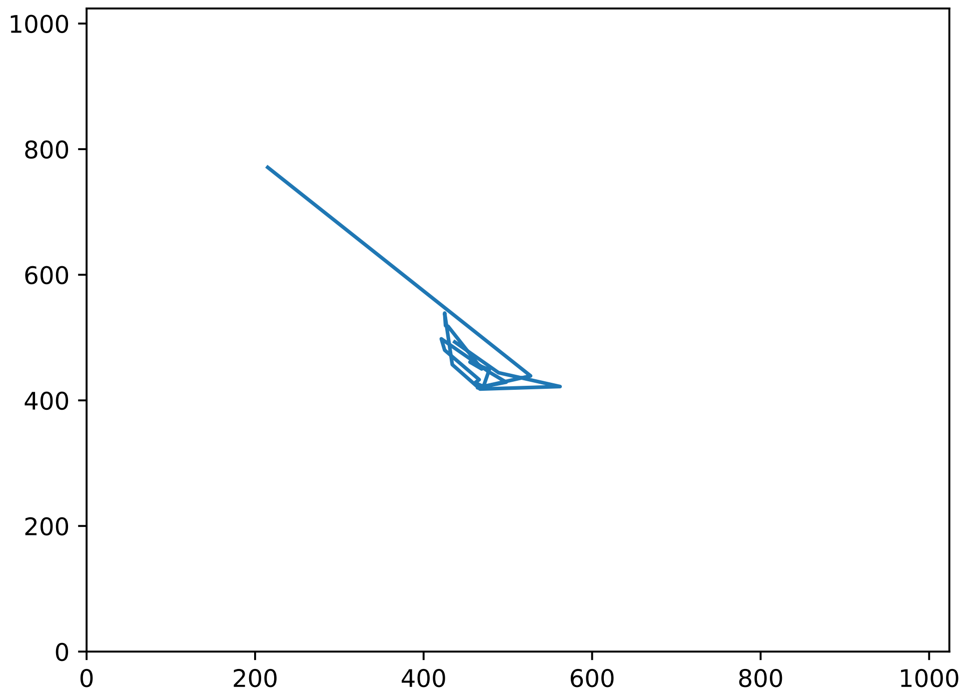


Figure 4.14: Plot of the different explored positions to localize the boat placed in (512,512).

stochastic computing, this phenomenon is called temporal dilution. Therefore, we optimized the architectures to avoid low probabilities by max-normalizing the likelihoods. This leads to a significant speed up of the computing.

To generate the stochastic bit streams for our machine, we chose to use Linear Feedback Shift Registers (LFSRs). LFSRs produce random numbers but they are part of a category called pseudo-random generators since their output is deterministic and can be predicted. However, for our applications, LFSRs generate good enough random numbers. Nonetheless, since LFSRs take a considerable space in our circuit, we would like to replace them by some other smaller random number generators in the future.

Until now, the Bayesian sampling machine has been used for applications where the searched variables have all the same cardinality. For example, in the boat localization example, both searched variables (X and Y) were discretized on the same number of values. Having different cardinalities for these variables would require simple changes in the control of the sampling unit of the Bayesian sampling machine.

Chapter 5

Audio foundations

The main goal of this thesis is to promote and develop alternative computing architectures. In the scope of this work, we presented two machines in chapter 4. Furthermore, in this project we aim to address two signal processing applications with our machines, namely the sound source localization and the sound source separation problem. Since this thesis mainly focuses on the tuning of these architectures and on their adaptation to the applications, this chapter's intention is to provide the necessary background knowledge of signal processing to understand the algorithms presented in this document. Moreover, a set of existing representative techniques from the state-of-the-art for both applications are described.

Note that the notations used in this section differ from the typical notations used by the community. The reason is that we aim to use a uniform notation across the whole document.

5.1 Sound Source Localization (SSL)

The first application that will be treated with our Bayesian machines is Sound Source Localization (SSL). This section is dedicated to SSL. There exist various different approaches to deal with this problem. The problem is defined before providing an overview of the methods proposed by the community. Since it is an old problem, there exists a lot of literature introducing this topic: [22], [43] and [113].

5.1.1 Task definition

The goal of sound source localization is to estimate the position of a sound source in a room (or another specific environment) given audio recordings from that source. A sound source can typically be a speaker or a person standing in the room. The signals are recorded using various microphones which can be put in pairs. These microphones can be placed across the room along the walls or in the corners depending on the setup. Setup with microphone arrays in the middle of the room (for example on a table) can also be found. Moreover, for some methods, it is crucial to place the microphones in pairs.

Although SSL is often seen as a signal processing problem, researchers from other fields also get involved. For example, neuroscientists try to understand how humans localize sound sources [111]. Historically, John William Strutt, 3rd Baron Rayleigh, was the first to study human sound localization. His *duplex theory* of human sound localization has been published in 1877 in [104]. It attempts to model the way humans localize sounds using their ears. It states that humans use the Inter-channel Time Difference (ITD) or the Inter-channel Level Difference (ILD) depending on the frequency of the sound. Below 1.5 kHz, the ITD would be used to localize a source. For sounds above 1.5 kHz, humans would use ILD to estimate a sound source position.

SSL can be performed in anechoic environment which means that there are no reverberations. In more realistic setups, reverberations are taken into consideration. Moreover, the number of present sound sources is also an important parameter. Single source localization aims to localize one sound source. Whereas multi-source localization deals with several sound sources that can be simultaneously active.

5.1.2 Basics of audio processing

In this section, we define the basic concepts of audio signal processing used in this work. Moreover, some commonly used features for sound source localization are introduced.

Fourier transform

This work is mainly done in the frequency domain. Therefore, the Fourier transform is used to transform a temporal signal into the frequency domain. More precisely, we use the Short-Time Fourier Transform (STFT), i.e. a sequence of Discrete Fourier Transforms (DFT) calculated using a sliding analysis window [95]. Given the microphone signals, which are sampled at sampling rate $f_s = 1/T_s$ with T_s the sampling interval we compute the STFT. In order to not deviate too much from existing signal processing conventions, i.e. capital letters for time-frequency domain whereas lower case designates signals in the time domain, we introduced the following notation for the Fourier transform in this document. The STFT of a signal m^1 from microphone 1 is noted with a calligraphic capital letter $\mathcal{M}^1$. For a specific frame l, the Fourier transform which is the DFT computed for that given frame is noted $\mathcal{M}^1_l$. Using this notation, the DFT for the signal m^1 is computed as follows:

$$\mathcal{M}^1_{k,l} = \mathrm{DFT}(m^1_{(lT_s)}) = \sum_{i=0}^{N-1} m^1_{((i+lH)T_s)} e^{-j2\pi\frac{ik}{N}} \tag{5.1}$$

where k, l are the frequency and time-frame indexes, N is the size of the analysis window, and H is the size of the window shift.

Signal processing

Since most of techniques for SSL use features based on two microphones, we will consider the microphone pair composed of microphones m^1 and m^2. The signal m^1_n captured by microphone 1 at sample n is represented by a linear convolution $*$ of the Room Impulse Response (RIR) from the source position to the microphone 1 position u^1_n with the signal emitted by the source s_n:

$$m^1_n = u^1_n * s_n = \sum_{i=0}^{Z} u^1_i \cdot s_{(n-i)} \tag{5.2}$$

where Z is the size of the analysis window.

To transpose the signal into the time-frequency domain for further analysis, the signal is approximated by:

$$\mathcal{M}^1_{k,l} \simeq \mathcal{U}^1_k \cdot \mathcal{S}_{k,l}. \tag{5.3}$$

When recording the signal with two microphones, we get two signals m^1 and m^2 defined as in Eq. 5.2:

$$\begin{aligned} m^1_n &= u^1_n * s_n = \sum_{i=0}^{Z^1} u^1_i \cdot s_{(n-i)} \\ m^2_n &= u^2_n * s_n = \sum_{i=0}^{Z^2} u^2_i \cdot s_{(n-i)} \end{aligned} \tag{5.4}$$

Once transformed into the frequency domain we obtain:

$$\begin{aligned}\mathcal{M}^1_{k,l} &\simeq \mathcal{U}^1_k \cdot \mathcal{S}_{k,l}\\ \mathcal{M}^2_{k,l} &\simeq \mathcal{U}^2_k \cdot \mathcal{S}_{k,l}\end{aligned} \tag{5.5}$$

To calculate $\mathcal{U}^1_k$ and $\mathcal{U}^2_k$ we use the Fast Fourier Transform (FFT) which is an implementation of the Discrete Fourier Transform (DFT):

$$\begin{aligned}\mathcal{U}_k &= DFT(u_n)\\ &= \sum_{i=0}^{N-1} u_i \cdot e^{-j2\pi ik/N}\\ &= |\mathcal{U}_k| \cdot e^{j\Phi_k}\end{aligned} \tag{5.6}$$

where Φ_k is the phase of the RIR.

For source localization, we use the Relative Transfer Function (RTF) needed to obtain the inter-microphone information from which several features can be extracted. The RTF is defined as:

$$\mathcal{R}_k = \frac{\mathcal{U}^2_k}{\mathcal{U}^1_k} \tag{5.7}$$

However, in practice, the RIR $\mathcal{U}^1$ and $\mathcal{U}^2$ are unknown since they are specific to the setup. Therefore, we approximate the RTF by using the microphone signals such as:

$$\frac{\mathcal{M}^2_{k,l}}{\mathcal{M}^1_{k,l}} \simeq \frac{\mathcal{U}^2_k \cdot \mathcal{S}_{k,l}}{\mathcal{U}^1_k \cdot \mathcal{S}_{k,l}} = \frac{\mathcal{U}^2_k}{\mathcal{U}^1_k} = \mathcal{R}_k \tag{5.8}$$

Features for sound source localization

As mentioned previously, sound source localization needs at least two microphones because some inter-microphone information is used. In practice, several pairs of microphones are placed at different positions in the room. In the literature, arrays of microphones are often used.

There are different features that can be deduced from the microphone pairs. These features can all be used in certain ways for source localization. Namely, the features are:

- Inter-channel Level Difference (ILD)
- Inter-channel Phase Difference (IPD)
- Inter-channel Time Difference (ITD)

For each of these features, at least two microphones need to be used. However, the precision of the localization increases with the number of microphone pairs when combining the features obtained from the different pairs of microphones. The distance between the microphones of a given pair is d. The features are based on the comparison of the two signals recorded by the microphones. Note that in the literature the word *inter-channel* is sometimes replaced by *inter-microphone* or *interaural*.

Inter-channel Level Difference (ILD)

The Inter-channel Level Difference (ILD) is the modulus of $\mathcal{R}_k$ defined in Eq. 5.7.

$$\mathrm{ILD}_k = |\mathcal{R}_k| = \frac{|\mathcal{U}_k^2|}{|\mathcal{U}_k^1|} \tag{5.9}$$

More precisely, in dB the ILD can be written as:

$$\mathrm{ILD}_k(dB) = 20\ \log_{10}|\mathcal{U}_k^2| - 20\ \log_{10}|\mathcal{U}_k^1| \tag{5.10}$$

Inter-channel Phase Difference (IPD)

The Inter-channel Phase Difference (IPD) defined as the phase of the RTF:

$$\begin{aligned}\mathrm{IPD}_k &= arg(\mathcal{R}_k)\\ &= arg(\mathcal{U}_k^2) - arg(\mathcal{U}_k^1)\\ &= \Delta\Phi_k\\ &= \Phi_k^2 - \Phi_k^1\end{aligned} \tag{5.11}$$

The IPD provides an angular information of the sources related to the microphone pair position. Note that this feature has been used in our work.

Inter-channel Time Difference (ITD)

The ITD measures the delay Δt_k between the arrival of the two sound waves to the different microphones. After sampling, since we deal with digital signals, the ITD becomes Δn_k.

$$\mathrm{ITD}_k = \Delta n_k = \frac{\Delta\Phi_k + 2\pi q}{2\pi\frac{k}{N}} \tag{5.12}$$

with $q \in \mathbb{N}$. Since the IPD can only be measured with a 2π accuracy, the ITD is impacted by this ambiguity, hence the $2\pi q$ factor.

Free field model

To solve the source localization problem, it is known that one can use a simplified acoustic model, namely the *free field model.* This model assumes that the reflections of the sound waves on every wall of the room have negligible effects, that all microphones are "floating" in the room and the sound waves are delayed and have the same attenuation since the microphones are located closely (d is small compared to the source to microphone distance).

Hence, the ILD feature is not usable anymore since

$$u^1_{n_1} \simeq u^2_{n_2} \Leftrightarrow ILD = 1.$$

This implies

$$\mathcal{R}_k = \frac{\mathcal{U}^2_k}{\mathcal{U}^1_k} \simeq e^{-j2\pi\frac{k}{N}(n_2-n_1)} \tag{5.13}$$

Notice that $\Delta n = n_2 - n_1$ does not depend on the frequency k contrary to $\Delta\Phi_k = 2\pi\frac{k}{N}(n_2 - n_1)$.

5.1.3 State of the art of source localization

As previously mentioned, Sound Source Localization (SSL) is a very old research topic since the first work has been published in 1877 in [104] by Rayleigh. Since then, many other researchers worked on it and published interesting results.

An important historical reference working on sound source localization is [71]. The authors propose a Time Difference Of Arrival (TDOA) estimation based on Generalized Cross Correlation(GCC). The TDOA is the time difference between the arrival time of the two microphone signals. It will depend on the frequencies with the highest energy. In this paper, a maximum likelihood estimator is presented to estimate the source location based on the time delay between the noisy signals recorded by the microphones.

Many probabilistic methods have been proposed inspired by the human binaural hearing [103], [123] and [37]. They use microphone pairs in order to use common features such as the IPD and the ILD, introduced in section 5.1.2.

In [80], the Model-Based Expectation Maximization Source Separation and Localization (MESSL) method is presented. It aims to localize and separate multiple sound sources from an under-determined reverberant two-channel recording. An EM algorithm [38] is used to compute the maximum-likelihood parameters which leads to localizing the multiple sources. As a byproduct of this method, masks are created which can directly be applied on the spectrogram to separate the source in a second step. Another example of a model-based localization using the EM algorithm is given in [89].

Introducing a new aspect in classical sound source localization, the authors of [40] intent to also analyze their proposed algorithms in terms of complexity, latency and memory needs. Note that this is totally in the spirit of our project. They propose two new Distributed Expectation-Maximization (DEM) algorithm which are the Batch-DEM (BDEM) and the Recursive-DEM (RDEM). In this case *distributed* means that the localization task is distributed on the different microphone pairs. They are both used in a distributed manner, hence their name. The BDEM processes the data after it has entirely been acquired. Whereas the RDEM constantly updates its result based on the current data. In terms of performance and also regarding the other criteria mentioned above, the RDEM achieved better results since it processes the data along the way.

In [76], the authors propose a multi-source localization method dedicated to noisy and reverberant environments. It is initially based on [40] but adds some interesting points. It uses Complex-Valued Gaussian Mixture Model (CGMM) where all components correspond to the possible potential source positions. To build robust binaural features they use a Direct-Path Relative Transfer Function (DP-RTF) used in [75] which is the ratio of the direct path of the acoustic transfer function between both binaural microphones. Here, it is extended to multi-source problems. Interestingly, instead of an Expectation Maximization (EM) algorithm, they opted for a more efficient method, the convex-concave optimization procedure. At the end, they use a simple peak selection procedure to select the multiple estimated source locations in the final distribution.

Another multi-source localization approach is presented in [58]. They work in Wireless Acoustic Sensor Networks (WASNs) where each node provides an Direction Of Arrival (DOA) estimation. The DOA is the direction from which the sound waves arrive to the microphones. The problem addressed by the author happens when two nearby sources are estimated as two separated sources and needs to be merged. Moreover, they work on DOA estimation error modeling. The method has a very low complexity which allows it to be real-time. However, the complexity measurements are done on estimating the CPU time, still a very high-level and inaccurate analysis.

All the work presented in this section is using the time-frequency signal representations. Although it is popular in the signal processing community to use the Fourier transform, it has some drawbacks. The approximation of equation 5.5 becomes better as the analysis window of the Fourier transform N increases. However, having a large analysis window generates a lot of processed data. The window size used to compute the Fourier transform can impact the overall performance of the system. When designing dedicated hardware for signal processing application such as in this project, the resources are very limited. Therefore, operating on small analysis windows allows to reduce the activity of the system and hence reduce the power consumption. One of such models is presented in [70].

5.2 Source separation

In this project, we aim to treat two different applications with our Bayesian machines. The second task, namely the source separation, is presented in this section as well as an overview of the current developed techniques. A lot of literature is providing a good overview on this research domain including several books [31], [126], [119] and [79].

5.2.1 Task definition

The goal of source separation is to recover the signals originally emitted by some sound sources and mixed together based on the observation of the resulting mixture signals. Generally, the sources are placed in a room and the mixtures are recorded by microphones also present in the room. The typical example of source separation is the cocktail party problem which was first analyzed by Colin Cherry at the beginning of the 20th century. Some people are talking at the same time in a room and another person attempts to focus on one speech signal emitted by a specific speaker. Although the human brain can easily handle this situation, the problem remains difficult for digital signal processing.

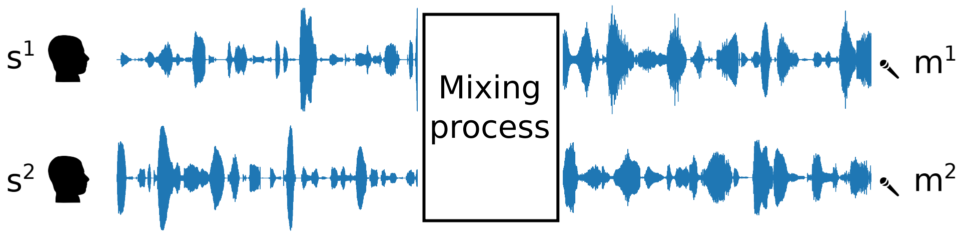


Figure 5.1: The mixing process of two sources recorded by two microphones.

Figure 5.1 illustrates the mixing process which defines the signals recorded by the microphones. Assume there are two sources, namely s^1 and s^2, speaking at the same time. Their signals are mixed according to the mixing process and the only information we obtain are the signals recorded by our microphones m^1 and m^2. The task consists of estimating the original source signals based on the recorded signals. Moreover, the mixing process which is typically specified by the mixing matrix A is also unknown.

A sub-area of source separation focuses on blind source separation (BSS). BSS is the separation of sources with very little information or without any information. Information could be the number of sources, the position of the sources, the mixing process. Let J be the number of sources to separate and I the number of microphones.

The difficulty of the problem may vary depending on several parameters. The dimensions are an important factor. There are 3 types of mixtures:

- Over-determined mixture: $I > J$. There are more microphones than sources.
- Determined mixture: $I = J$. There are as many microphones as sources.
- Under-determined mixture: $I < J$. There are less microphones than sources.

The under-determined case is the most difficult.

5.2.2 State of the art of source separation

Source separation is a vast research area. Several books and papers have been published over the years. In 2009, already 22,000 papers were recorded by Google Scholar [31]. Now, there are more than 1,480,000 results when searching for Blind source separation techniques. Several approaches exist which are using multiple different mathematical tools and provide solutions depending on different type of assumptions. However, this domain is still very much in progress. Due to the high number of techniques and publications, this present section is nothing but a short introduction to source separation.

There exist different mixture models which define the mixture process. There are two main mixing models. The first one is the *instantaneous model*:

$$m_t^i = \sum_{j=1}^{J} a^{(i,j)} \cdot s_t^j + b_t \tag{5.14}$$

where $a^{i,j}$ is an element of A which is the mixing matrix. b_t is the microphone noise at time t. The second one is the *convolutive model*:

$$m_t^i = \sum_{j=1}^{J} \sum_{\tau} a_\tau^{(i,j)} \cdot s_{t-\tau}^j \tag{5.15}$$

where $a^{(i,j)}$ is a Finite Impulse Response (FIR) filter. In other words, the model is performing a linear time-invariant filtering. The convolutive model is more suitable for indoor audio tasks with reverberations than the instantaneous model.

Although the mixing process of the sources is performed in the time domain, most of the separation techniques work in the time-frequency domain. They take advantage of the parsimonious representation of sources in the time-frequency domain which facilitates the separation. In the TF domain, few source coefficients have a lot of energy and therefore the sources overlap less than in the time domain.

The audio source separation methods can be categorized into three main families:

- Independent Component Analysis (ICA)
- Binary Masking (BM) and Computational Auditory Scene Analysis (CASA)
- Probabilistic Model (PM) and Bayesian inference

In the following, an overview of these three groups of techniques is provided. Note that these groups are not exclusive. There are many approaches which could be affiliated to more than one family.

5.2.3 Independent Component Analysis

Independent Component Analysis (ICA) is one of the first techniques which has been developed for source separation. The method is to apply a matrix on the mixture which separates the signals and provides the independent source signals [22]. It assumes the (pair-wise) independence of the source signals [64]. Using these filters allows to select one source, to enhance it, and attenuate the other, based on their positions. Moreover, one has to consider that sources generate echoes as well which need to be removed from the mixture [126, Chapter 14] and [119]. This technique is also called beamforming. Therefore, one has to obtain the Direction Of Arrival (DOA) before applying beamforming techniques. The estimation of the DOA is more a localization task which has been presented in the first part of this chapter.

Some localization algorithms are very powerful to detect the DOA of multiple sources such as [92]. However, they perform well in an echo-free environment but their efficiency decreases operating in real-world environments.

There are different types of beamformers. Some simple fixed approaches are the delay-and-sum beamformer, the null beamformer [22]. They need a strong knowledge of the source DOA otherwise the source which should be attenuated can be amplified by error. Other approaches have been proposed which are more flexible and adapt to the recorded data [119]. However, all these approaches still require a precise information of the target DOA which in real-world environments can be inaccurate [119].

Therefore, some other approaches exist which are based on the entire mixing matrix A. Hence, if one is able to estimate the unmixing matrix $W = A^{-1}$, one can easily apply W to the recorded signals. However, in this case, the mixing matrix needs to be invertible (which is not the case in under-determined mixtures). Also, to apply these techniques, one needs to assume that all sources are independent. In Frequency-Domain Independent Component Analysis (FDICA) one considers one mixing matrix A for each frequency. Some work has been published which is based on the assumption that, for a given frequency bin, signals in the sources are present in several time-frames, i.e. [109] and [118].

Unfortunately, beamforming and FDICA are methods which work well for over-determined or determined setups, when the number of sources is equal or smaller than the number of channels [119].

5.2.4 Binary Masking

Another type of method that exists for source separation is called binary masking. The concept is to apply masks on the time-frequency representation of the recorded signal to unmix the source signals. The masks can be binary or continuous masks. Each mask is specific to one given source and hence represents its time-frequency profile. The masks are each applied on the spectrogram to obtain the desired source signal. At the end, one obtains J channels with each a single source. This method is suitable for under-determined mixtures, where $I < J$. It is the most difficult configuration since the mixing matrix is not invertible.

Figure 5.2 shows the different steps of the binary masking approach. The figure is taken from [13]. Note that this method assumes that the sources are sparse to ensure that only one source is dominant in each time-frequency frame. In [105] the authors present a real-time implementation of binary masking.

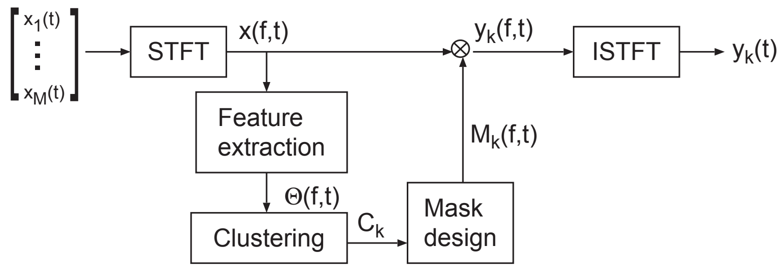


Figure 5.2: Flow of the binary masking method. Figure taken from [13].

The most famous method of binary masking is called Degenerate Unmixing Estimation Technique (DUET) [125]. It assumes a 2-channel anechoic mixture and a single source present in each time-frequency bin. The separation has a fair quality with respect to the simplicity of the method. It is representative of other more complex time-frequency binary masking methods, i.e. [106] and [80]. However, it is limited to anechoic conditions and is not robust to reverberations.

The sparse component analysis takes advantage of the sparsity of the signals, hence it is not a completely BSS method due to *a priori* knowledge. In case of sparse sources, scatter plots provide a clear information which helps to estimate the mixing matrix. The method of Bofill and Zibulevsky [21] is using the above mentioned method. It assumes at most two sources present in each time-frequency bin and a 2-channel convolutive mixture. It estimates the mixing matrix A using scatter plots followed by clustering. Once A has been estimated, a minimal L_1-norm representation of the sources is computed. It is a kind of shortest path estimation of the source signal. This method provides poor separation quality and unfortunately it is not robust to reverberations.

More recently deep-learning based methods were developed which use Deep Neural Networks (DNNs) to estimate the binary masks. An overview of the methods using neural network for source separation and speech enhancement is provided [121].

5.2.5 Probabilistic Models for source separation

A third large family of techniques for sound source separation is made of the Bayesian approaches. They use a probabilistic model to perform Bayesian inference to estimate the mixing matrix A and simply model it as a random variable in the probabilistic model. This allows to address cases where A is not invertible. Moreover, it enables to integrate *a priori* knowledge on the sources. Typically, these approaches make an extensive use of the EM algorithm [38]. The sources are modeled as complex Gaussian distributions. This was originally proposed in [44] and [17]. The sources are assumed to be mutually independent. They are also assumed to be individually independent across frequency bins and time frames.

Since in general there is no closed-form solution for typical machine learning solutions, one uses the Expectation-Maximization (EM) algorithm [38]. Furthermore, the EM algorithm is often used in combination with non-negative matrix factorization (NMF) technique to model the variance of the source signals and thus reduce the dimension of the problem. NMF is frequently used in audio processing [74]. It is especially used to model the time-frequency power distribution [50].

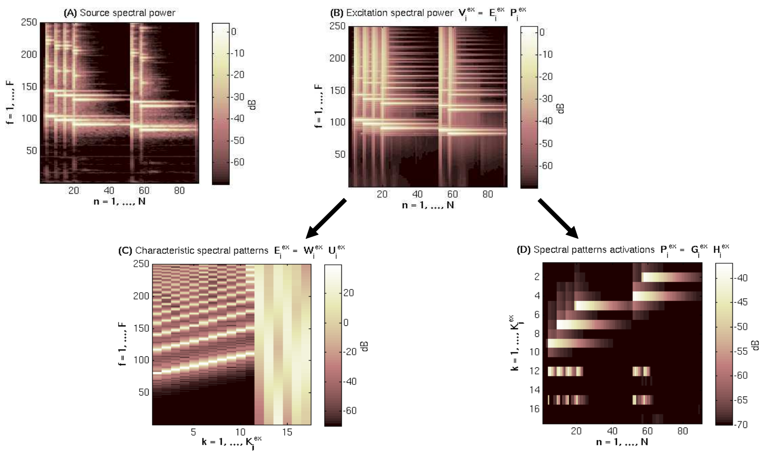


Figure 5.3: NMF example for xylophone notes. Figure taken from [97].

Figure 5.3 shows an example of NMF for a spectrogram of several xylophone notes. The NMF decomposition allows to decompose the variance (or power spectral density) of the source signals in the time-frequency domain into two matrices: the characteristic spectral pattern (spectrogram C in the figure) and the spectral patterns activation (spectrogram D in the figure).

A first example of an EM algorithm which estimates the sound source signals using a Gaussian source model combined with NMF is presented in [96]. Here, the authors are modeling the source components corresponding to the NMF decomposition of the source variance. The components belonging to the same source are mixed with the same filter. It assumes a convolutive mixture. Another example is given in [41]. This work aims to model setups with reverberations for under-determined mixtures.

Moreover, some authors augment the NMF method by an additional feature which leads to an array of matrices, called a tensor [114]. In this case, they added the directionality of the sound into the NMF. Then, the NMF method becomes the Non-negative Tensor Factorization (NTF) method. This work has been developed by a company called Analog[1]. They implemented the approach presented in [114] on a Digital Signal Processor (DSP) to perform source separation on low-power devices. This could have been a good benchmark for our work. Unfortunately this was developed internally and no details were publicly released, hence we have not been able to use it.

Overall, one can say that the Bayesian approaches are excellent to insert *a priori* knowledge [97]. For example, the Computational Auditory Scene Analysis (CASA) approaches use informations extracted from image or video [120]. These features are then added to the Bayesian model for source separation. However, the performance of the EM algorithm highly depends on its initialization. The initialization of the EM approaches is still a crucial problem which remains unsolved for reverberant under-determined setups.

For example in [89] the authors propose a CASA-EM system. In a first step, the DOA of the sources is computed using the ILD and ITD features. Each source present in the mixture is described using a Gaussian representation. Then, an EM algorithm estimates the weight, mean, and variance of each sources representation in order to apply a Gaussian filtering algorithm to separate the sources. This work is a great example of a simultaneous localization and separation method.

5.3 Discussion

The intention of this chapter was to provide an introduction to audio signal processing and define the concepts needed for the upcoming chapters. The sound source localization (SSL) problem has been defined. An overview of the different existing approaches in literature dealing with SSL has been described. Moreover, the source separation task has been stated and the most common methods have been presented.

Researchers someday started with looking at the human ear in order to imitate it and building an artificial way to localize sources. This led to a broad use of the Fourier transform as this mathematical tool is modeling the cochlea, the part in the human ear which performs frequency filtering. The cochlea can be described as a biological Fourier analyzer [111]. However, is this mathematical tool, which we are so extensively using, really exactly

[1]Analog devices: `https://www.analog.com/`.

doing what humans do by default? One could argue that we frequently use the Fourier transform as it is the only tool we were able to develop in order to keep up with nature. Especially, when thinking about simple systems, one might have to rethink the use of the Fourier transform twice. Computing the Fourier transform always adds some time, resources and energy required by the system. Since we aim to build low-power system we really asked ourselves if we need the Fourier transform and showed that in the temporal domain there are still some opportunities.

Chapter 6

Mono-Sound Source Localization in the time-frequency domain

In this chapter, a first sound source localization (SSL) method will be presented. It estimates the position of one sound source located in one room and works in the time-frequency domain. This means that the Fourier transform is necessary as a pre-processing step.

This concept was presented in an conference paper at the 2017 IEEE International Conference on Rebooting Computing (ICRC) in Washington, USA [46]. Further experiments were presented at the 2018 IEEE International Conference on Cognitive Informatics & Cognitive Computing in Berkeley, USA [47].

The overall layout of this chapter is as follows. First, the probabilistic model and the needed signal processing are described. Second, the Bayesian machine adapted to this application and its implementation in VHDL are introduced. Third, the simulated experiments and real world experiments and their respective results are presented.

6.1 Signal pre-processing

In order to set up the probabilistic model used for the SSL, we need to define a few basic concepts used in the model. Moreover, the signal is preprocessed before we can start to infer on the model.

The probabilistic model is based on the Inter-channel Phase Difference (IPD), which was introduced in section 5.1.2. It is a feature commonly used for sound source localization. The IPD provides an angular information. Therefore, microphones need to be placed in pairs in the room with a fixed inter-microphone distance. As each microphone pair gives the direction of the sound source, this model uses several microphone pairs to fuse all their angular informations using the Bayesian machine described in 4.2.

The free field model is assumed as acoustic model. This means that the microphones are omni-directional and fixed on stands that have no effect on the sound propagation (in other words, all microphones are "floating" in the room). Also, the reverberations on the walls are assumed to be negligible, and the distance from the source to the microphone is large enough to consider the acoustic waves reaching the microphones as plane waves.

For the sake of simplicity, let us first look at one microphone pair formed by microphone 1 and 2. The values recorded at time t on microphones 1 and 2 are given by m_t^1 and m_t^2. The signal emitted at time t by the sound source is given by s_t.

Due to the free field model, both microphone signals are attenuated and delayed versions of the source signal s_t , and m_t^2 is a delayed version of m_t^1:

$$\begin{aligned} m_t^1 &= a \cdot s_{(t-t_s)}, \qquad \text{with} \quad 0 < a < 1, \\ m_t^2 &= m^1_{(t-t_0)}. \end{aligned} \tag{6.1}$$

The delay t_s represents the delay due to the distance L between the source and the microphone m^1. The delay t_0 corresponds to the wave path difference between the two microphones (we assume that the attenuation on this part of the wave path is negligible). It depends on the azimuth θ of the source, which is defined as the angle between the axis perpendicular to the inter-microphone axis and the source direction (see figure 6.1). Assuming the source-to-microphone distance L much larger than the the inter-microphone distance d, we have:

$$t_0 = \frac{d}{v}\sin(\theta), \tag{6.2}$$

where v is the speed of sound ($\simeq 340$ m.s^{-1} in the air). Therefore, a measure of t_0 (or an equivalent information) can lead to an estimation of the source azimuth. Merging the azimuth information provided by several microphone pairs (at least two) can then lead to an estimate of the absolute source position using the probabilistic model explained in the upcoming section.

To this aim, the microphone signals are sampled at sampling rate $f_s = 1/T_s$ with T_s the sampling interval and we calculate their Short-Time Fourier Transform (STFT), i.e. a sequence of Discrete Fourier Transforms (DFT) calculated on a sliding analysis window [95]. In order to not deviate

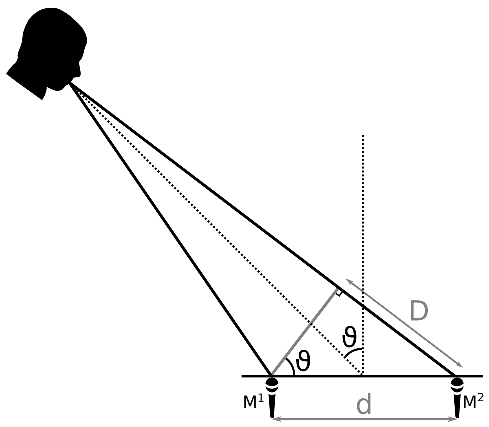


Figure 6.1: Schema of the source-to-microphones wave propagation.

too much from existing signal processing conventions, i.e. capital letters are used for the time-frequency domain whereas lower case designates signals in the time domain. We introduced the following notation for the Fourier transform in this document. The STFT of a signal m^1 is noted with a calligraphic capital letter $\mathcal{M}^1_{k,l}$ where k is the frequency index and l the time-frame index. Using this notation, the Fourier transform of the signals m^1 and m^2:

$$\begin{aligned}\mathcal{M}^1_{k,l} &= \mathrm{STFT}(m^1_{(lT_s)}) = \sum_{i=0}^{N-1} m^1_{((i+lH)T_s)} e^{-j2\pi\frac{ik}{N}} \\ \mathcal{M}^2_{k,l} &= \mathrm{STFT}(m^2_{(lT_s)}) = \sum_{i=0}^{N-1} m^2_{((i+lH)T_s)} e^{-j2\pi\frac{ik}{N}},\end{aligned} \tag{6.3}$$

where k, l are the frequency and time-frame indexes, N is the size of the analysis window, and H is the size of the window shift. Inserting model (6.1) into (6.3), we obtain:

$$\mathcal{M}^2_{k,l} \simeq \mathcal{M}^1_{k,l} e^{-j2\pi\frac{kt_0}{NT_s}}. \tag{6.4}$$

Eq. (6.4) is an approximation mainly because of the finite size of the STFT window, but if model (6.1) holds, (6.4) is a very good approximation in particular for STFT bins that contain a significant amount of energy.

Furthermore, we define a grid of 2D source positions to discretize the room. Usually we use 8×8 or 32×32 or 64×64 grids. The coordinates of the sound source estimated by this model are represented by the probabilistic variable $\boldsymbol{C}$ which is a conjunction of two variables C_x and C_y. For every candidate source position (c_x, c_y), we calculate the corresponding source azimuth $\theta_m(x, y)$ with respect to each microphone pair indexed by m with a inter-microphone distance d. The corresponding (theoretical) candidate delay is given by (6.2) with $\theta = \theta_m(x, y)$. Using (6.4) the corresponding (theoretical) inter-channel STFT coefficient ratio $R^m_{(k,l)}$ for microphone pair m is given by:

$$R^m_{(k,l)} = \frac{\mathcal{M}^{(2,m)}_{(k,l)}}{\mathcal{M}^{(1,m)}_{(k,l)}} \simeq e^{-j2\pi \frac{kd}{NvT_s} \sin(\theta_m(x,y))} \tag{6.5}$$

From (6.5), we can extract the theoretical IPD $\alpha^m_{(k,l)}$ which depends on the a specific frequency k and the position of the cell (x, y):

$$\alpha^m_{(k,l)}(x, y) = \arg(R^m_{(k,l)}) \simeq -2\pi \frac{kd}{NvT_s} \sin(\theta_m(x, y)) \tag{6.6}$$

Note that here, $\alpha^m_{(k,l)}$ is not depending on the time-frame index l. It only depends on the position of the source (x, y) and the frequency bin k. Hence, we can write the theoretical IPD as:

$$\alpha^m_k(x, y) = \arg(R^m_k) \simeq -2\pi \frac{kd}{NvT_s} \sin(\theta_m(x, y)). \tag{6.7}$$

6.2 Probabilistic model

In this section we introduce the probabilistic model used for the SSL method. Therefore, we first need to defined the probabilistic variables:

- $\boldsymbol{C} = (C_x, C_y)$: the probabilistic variables associated to coordinates of the sound source with $C_x \in [0, \text{dim}_y]$ and $C_y \in [0, \text{dim}_y]$ with dim_x and dim_y the dimensions of the room in the x and y axis. Both variables are discretized on respectively dim_x and dim_y values. The cardinality of this probabilistic variable is $w = card(C_x) \times card(C_y)$.
- $\boldsymbol{\phi} = \{\phi^m_{k,l}\}$: the set of probabilistic variables associated to the measurement of each IPD for each microphone pair m with $m \in \{1, 2\}$, each frequency bin $k \in \{1, K\}$ and each time frame $l \in \{1, L\}$. The value of the IPD is an angular information discretized in the interval $[0, 2\pi[$.

Having specified the probabilistic variables, we can defined the decomposition on the probabilistic model. Knowing the localization of the source, the IPD variables are independent and since conditioned on $\boldsymbol{C}$ the $\boldsymbol{\phi}$ are independent, the decomposition on $\boldsymbol{C}, \boldsymbol{\phi}$ is written as follows:

$$P(\boldsymbol{C}, \boldsymbol{\phi}) = P(\boldsymbol{C})P(\boldsymbol{\phi}) = P(\boldsymbol{C}) \prod_{m,k,l} P(\phi^m_{k,l}|\boldsymbol{C}) \tag{6.8}$$

Since we do not have any prior knowledge on the localization of the source, we model $P(\boldsymbol{C})$ as the uniform distribution.

Given an acoustic source emitting from position (x, y), $\phi^m_{(k,l)}$ is assumed to follow a Gaussian distribution centered on α^m_k (given by (6.6)) and of variance σ^2_ϕ:

$$\phi^m_{k,l} \sim \mathcal{N}(\alpha^m_k(c_x, c_y), \sigma^2_\phi). \tag{6.9}$$

Note that the choice of having a variance that is independent of k and l and c is just a simplifying assumption, which does not prevent this model to work well. It would be possible to adapt the variance to the different values of c. For the hardware implementation it is better to have the same variance for all the values of k, l and c as it will be described later in this chapter. In practice, (6.6) and thus (6.9) hold for all STFT bins that contain significant signal energy. We thus have a large set of STFT ratios that inform about the source location. However, since $\varphi^m_{(k,l)}$ is a phase measure calculated from sensor signals, it actually consists of a principal angle value within $[0, 2\pi[$. To ensure that (6.6) and (6.9) are not spoiled by phase ambiguity, a simple solution consists in i) setting d to a small value to minimize t_0, and ii) given d and other parameters, selecting the low-frequency bins for which t_0 is assumed to be less than a period of the spectral component,[1] i.e:

$$\frac{kd}{NCT_s} \ll 1 \Leftrightarrow k \ll \frac{NCT_s}{d}. \tag{6.10}$$

In practice, we set $d = 5$ cm and (6.10) is verified for a large range of frequency bins k.

In summary, $P(\phi^m_{(k,l)}|(c_x, c_y))$ is modeled as a normal distribution

$$P(\phi^m_{(k,l)}|(c_x, c_y)) = \mathcal{N}(\alpha^m_k(c_x, c_y), \sigma^2_\phi) \tag{6.11}$$

where $\alpha^m_k(c_x, c_y)$ is the theoretical mean value of the IPD for a source located at (c_x, c_y) and σ_ϕ the precision of the measure of the IPD. Our probabilistic

[1]Another solution would be to use a circular distribution such as the von Mises distribution instead of (6.9), but in the present study it is very easy to ensure (6.10) and thus using (6.9) is fine.

model for SSL consists of a series of distribution values defined as:

$$P(\varphi^m_{(k,l)}|c_x, c_y) = \frac{1}{\sqrt{2\pi}\sigma_\phi} \exp\Big(- \frac{(\varphi^m_{(k,l)} - \alpha^m_k(c_x, c_y))^2}{2\sigma^2_\phi}\Big) \tag{6.12}$$

They are conditioned on source position (c_x, c_y) through (6.6), and evaluated i) for each point of the $w = card(C_x) \times card(C_y)$ source position grid, ii) for a series of low-frequency TF bins where the sensor signals are assumed to have significant energy and (6.10) holds.

Given the signal recorded by the microphone pair m, we can compute $\varphi^m_{(k,l)}$ using the STFT for all m, k, l using equations 6.4 and 6.5 and use the evidences $\boldsymbol{\varphi}$ from equation 6.11 to compute the posterior probability distribution over $\boldsymbol{C}$ using:

$$P(\boldsymbol{C}|\boldsymbol{\varphi}) = \frac{1}{Z} \prod_{m,k,l} P(\varphi^m_{(k,l)}|\boldsymbol{C}) \tag{6.13}$$

In the next section, we describe how to evaluate this expression with a Bayesian Machine (BM). The above defined values of 6.12 will from now on represent the likelihoods of the Bayesian model for the naive fusion. Because the BM uses probability values corresponding to discrete variables, in our practical implementation the measured IPDs $\varphi^m_{(k,l)}$ are actually quantized (with a resolution that is appropriate for the SSL problem), and values of the continuous distribution 6.12 are turned into probability values.

6.3 Bayesian machine adapted to SSL

Once the probabilistic model was validated using high-level simulations in python, we focused on the development of the adapted hardware. Since the sampling space, i.e. the grid size place in the room, is reasonably small, we used the parallel Bayesian machine for exact inference described in section 4.2 to compute the inference.

The machine was adapted to the current application. To remember the global architecture of the Bayesian machine figure 6.2 shows the organization of the machine in a simplified manner. One can recognize the matrix-wise organization of the core. As explained in the section 4.2, each column of the machine represents a sensor which provides an additional information to take into consideration for the fusion. Note that the word *sensor* might be confusing because it does not automatically mean a real hardware sensor. For the current application, each column of the machine takes into account the

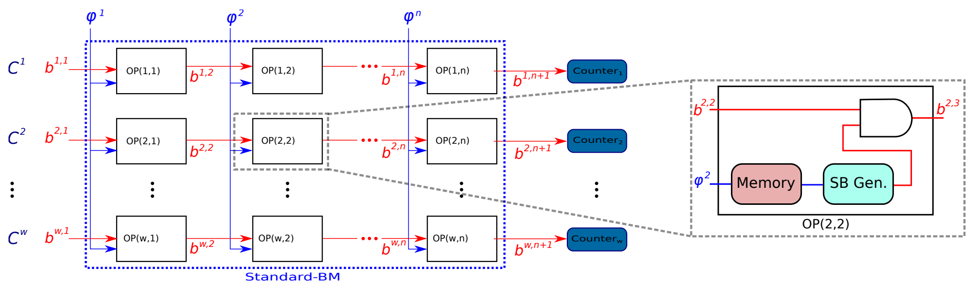


Figure 6.2: Architecture of the Standard-BM including a zoom on a single OP block of the machine. Bit streams are represented by red arrows. Blue arrows illustrate fixed-point numerical values.

measured Inter-channel Phase Difference (IPD) $\varphi^m_{(k,l)}$ of a given microphone pair m and a given time-frequency bin k in the current time frame l. The corresponding likelihood for the column is computed as defined in the previous section in equation 6.12. In practice, for a single frame, we take 50 different frequency bins which approximately represent the frequencies between 200Hz and 1000Hz. We focused on this interval since human voice is mainly active between these two frequencies. Since we have two microphone pairs, we obtain 100 columns: 50 frequency bins from microphone pair 1 and 50 more frequency bins from the microphone pair 2. In the machine, the columns are organized in pairs. This means that for each frequency bin, the IPDs will be of both microphone pairs will be consecutive.

Regarding the lines of the machine, each line represents one possible realization value of the searched variable whose probability distribution needs to be computed: the coordinates of the source: (c_x, c_y). In all the lines, the machine computes in parallel the probability distribution over the searched variable S. In our case, each line is responsible for a cell in the grid placed in the room in which the sound source localization runs. The machine has $w = card(C_x) \times card(C_y)$ lines. Typically, the coordinates are discretized into a grid of 8×8 or 32×32.

In practice, before performing the inference on the BM, the likelihoods are precomputed as defined in equation 6.12 and stored in the memory blocks in each OP block as shown in figure 6.2.

Due to the high number of columns (100 columns) in our application, we noticed that the low probabilities slow down the stochastic computation since the number of "1"s in the stochastic bit stream decreased drastically after a few columns. This phenomenon is called temporal dilution. Therefore, to tackle the temporal dilution problem, we improved the existing BM architecture to perform several times a max-normalization to speed up the computation.

6.3.1 Improvement of the Bayesian Machine: BM-sliced

Based on the previous considerations, we propose an improved BM, so-called the BM-sliced. This architecture was first presented in a conference paper at the 2017 IEEE International Conference on Rebooting Computing (ICRC) in Washington, USA [46].

The architecture of the BM-sliced is shown in figure 6.3. It is composed of multiple standard Bayesian machines (described in section 4.2) of limited number of columns, called slices, with a re-sampling unit between each consecutive slices. Figure 6.3 presents a BM-sliced with two slices. In the experiments, we will run a BM-sliced with 10 slices to solve our SSL problem.

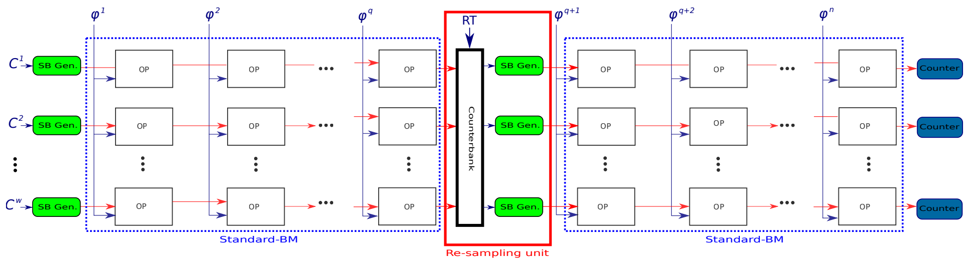


Figure 6.3: Architecture of a BM-sliced with two slices of $q = n/2$ columns each and 1 re-sampling unit in between.

To limit the temporal dilution, we apply the max-normalization described in section 4.2.5 over all probability distributions. The max-normalization allows to have at least one line per column with only "1"s as input which maximizes the probability of having a "1" at the corresponding output and thus accelerate the stochastic signal propagation. However, when the number of evidences, and hence the number of columns of the matrix, becomes large, the temporal dilution problem is still present.

The concept of dynamic re-sampling is used to tackle this problem. It consists in regenerating the stochastic signal after a subset of groups of evidences (i.e. here after a slice). In the same way that the Standard-BM uses an output counter bank to store the samples of the target distribution, the BM-sliced uses a counter bank in each re-sampling unit to regenerate signals with more "1"s. To implement the max-normalization in each re-sampling unit, as shown in figure 6.4, we set a re-sampling threshold (RT) value for all counters. When a counter reaches this value, the machine

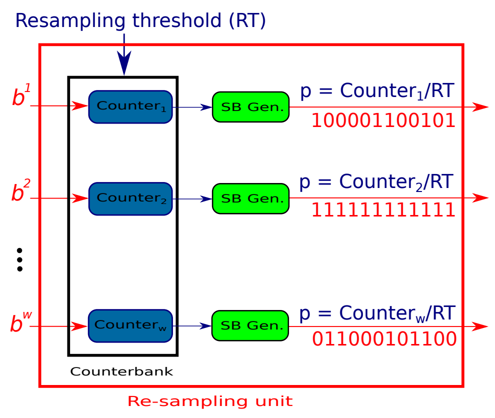


Figure 6.4: Re-sampling unit showing and example of max-normalization with the line 2 being at the maximum.

activates the process for the next slice, with prior probability

$$p_j = \frac{counter_j}{\text{RT}}$$

for line j.

The re-sampling unit maximizes and normalizes the output probability of a slice, and hence maximizes the number of "1"s in the stochastic bit stream (useful information) as input of the next slice. In other words, the re-sampling unit performs a max-normalization as described in section 4.2.5. We illustrate this particular case of a line reaching the threshold value in figure 6.4: the stochastic bit stream generated in line 2 is composed of only "1"s since line 2 is the first one with a counter having reached the RT value.

The overall effect of this dynamic re-sampling is shown in the experiment section.

6.4 Implementation on FPGA

Now that the overall architecture of the machine has been described in the previous sections, we will now focus on the implementation of the Bayesian machine in VHDL (VHSIC Hardware Description Language). Using this VHDL implementation, we were able to run the Bayesian machine on an FPGA (Field-Programmable Gate Array) and get more realistic experimentations. First, the implementation of the Bayesian machine in VHDL for SSL will be described. Second the on-chip likelihood computation process used on the FPGA is presented.

Note that this part is kept as generic as possible. Therefore, the figures showing the machine, as kept as generic as in section 4.2. The searched variable S which is C in the case of the SSL application. The known variable K (also called the observation of the model) are the phase difference ϕ in the SSL application presented in this chapter.

6.4.1 Circuit design in VHDL

Our Bayesian machine was implemented in VHDL to make more realistic experiments on FPGA and also to perform ASIC simulations to estimate the power/energy consumption of the architecture.

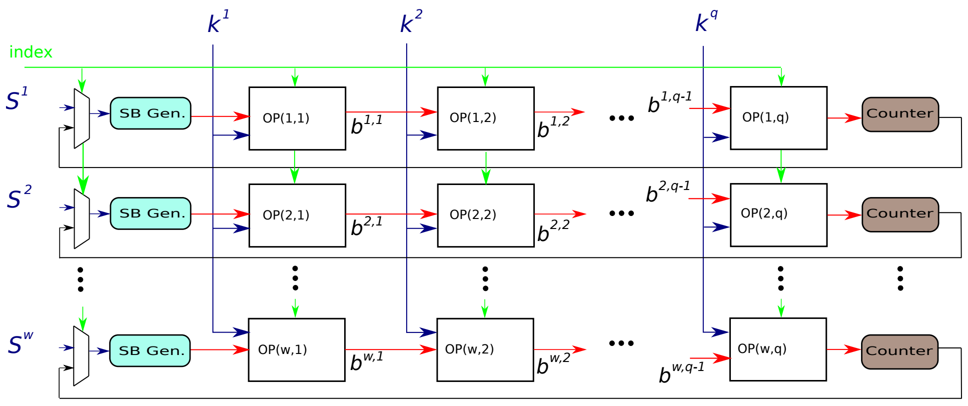


Figure 6.5: Filter-like implementation of the machine for the VHDL design.

When creating an electrical circuit, it is essential to analyze the length of the critical path in order to keep it as short as possible to be able to run the circuit at a faster pace. Also, the smaller the circuit is, the cheaper it will be to produce. A more filter-like architecture was implemented to obtain a smaller circuit. Basically, only one slice of the machine was coded while finite state machines (FSM) control the inputs of the implemented slice to adapt it to the other slices of the real machine. A simplified diagram of the circuit can be seen in figure 6.5. Due to the "filter"-like architecture, one can see that the output of the counters does not go to the next slice but is used as the input of the same slice (black line). An additional "index" wire controls the index of the current slice. The operator OP was adapted to this implementation and will be described more precisely later in this section.

The machine was split into columns for better modulability as shown in figure 6.6. The prior block generates the stochastic bit stream for each line. Note that the prior block provides the uniform distribution when the machine is in the first slice and then the values of the counters in all the upcoming slices.

Each column has a certain number of OP (operator) blocks which each mainly contain the AND-gate for the multiplication. In our circuit each column shares the same Random Number Generator (RNG). This means that the same random bit is shared among all OPs of a same column. Note that the input bit streams of a give AND gates are still independent since they are generated using two different RNGs. Since RNGs are circuits taking a lot of place and energy, we reduced the number of RNGs in order to decrease the circuit size.

The random bit can be shared among a column since, mathematically speaking, each line corresponds to a parallel independent sampling of the searched variable $\boldsymbol{C}$ which relates to the grid positions. However, it is important to have independent random bits in a same line as we perform stochastic multiplication. Each line samples the search distribution for a specific position $\boldsymbol{C} = c_j$. Due to this optimization, we considerably reduced the number of RNGs needed in the machine. Previously, we used $m * (p + 1)$ RNGs with p the number of columns and m the number of lines. Currently, only $p + 1$ RNGs are required as we need one per column of the machine and one for the prior column.

In the BM-sliced, the main improvement was to perform re-sampling between each slice to obtain a faster computation time and decrease the temporal dilution problem. The re-sampling process described in figure 6.4 is done by taking the counter value of the previous slice and simply give them as input to the priors of the next slice. The counter values describe the probability distribution computed until the end of the previous slice. Since the counters are encoded in the same bit width *nbit_prob* than the probabilities in the prior, passing the counters values to the priors performs the max-normalization. The probabilities are encoded between 0 and $2^{nbit\_prob} - 1$ where 0 is the lowest possible probability value and $2^{nbit\_prob} - 1$ the highest value: 1.

Note that each column possesses its own control block which communicates with the global control block. The action of the global control block can be illustrated by a Finite State Machine (FSM) which is provided in figure 6.7. When the machine receives the *start_init_gfsm* signal, it starts the memory initialization in order to calculate the likelihoods needed for the current slice. The exact procedure will be described later. Once this phase is over, the computation for the slice can begin. Once one counter has reached the re-sampling threshold (RT), the computation of the current slice ends and the counters are stored and given to the priors for the next slice. This is

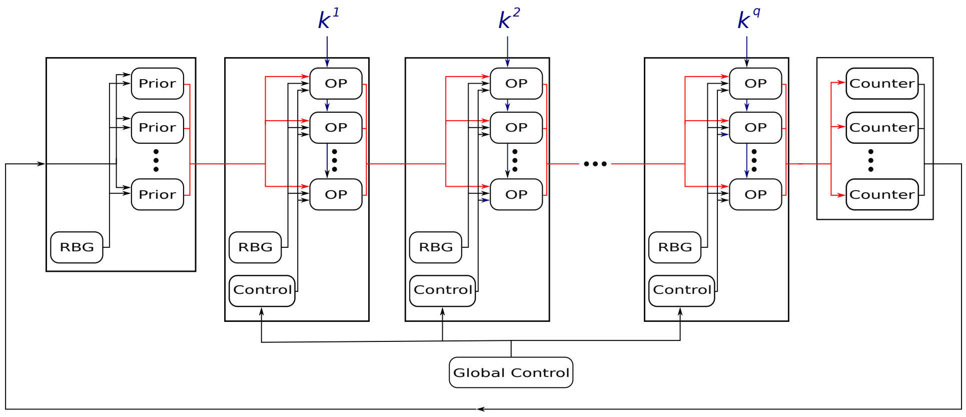


Figure 6.6: The simplified architecture of the VHDL implementation of the BM-sliced for the SSL.

repeated until the last slice is reached. At the end of the computation, the machine returns the counter values and turns back into idle state.

6.4.2 On-chip likelihood computation

When designing an electrical circuit of a computing architecture, one has to keep in mind where to place the memory and which type of memory to be used. Memory close to the computation module is very expensive but it is much more energy efficient to access data in SRAM than in DRAM. In fact, SRAM is several orders of magnitude more energy-consuming than memory accessing data in DRAM [98].

In CPUs nowadays the storage of the data in memory take up to half of the energy consumption of the global processor [63]. Therefore, it is crucial to reduce the memory need of an architecture when designing it. In other words, our Bayesian Machine still needs a lot of memory. In this section we present a memory optimization for our application.

Figure 6.8 shows the OP block used in the machine. One can see the memory block which stores the probability values of the max-normalized likelihoods that were pre-calculated. It allows to use the right probability depending on the incoming sensor value. The required size of memory is proportional to the discretization of the sensor value in order to have the corresponding likelihood value for each possible sensor value. When working with sensors having a fine discretization, the memory in the BM quickly increases. Hence this architecture is not scalable due to its tremendous required amount of memory.

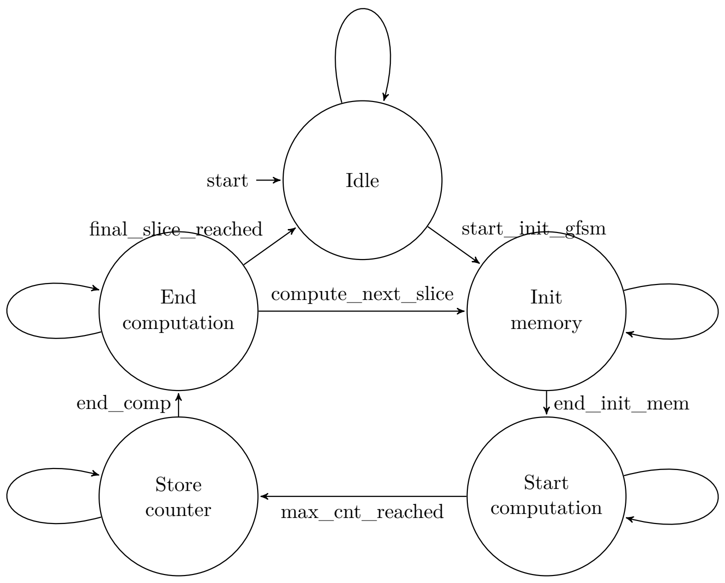


Figure 6.7: The automata of the global FSM of the BM.

For the SSL application, the memory usage is to high when running the circuit on an FPGA. Therefore, a memory optimization method is required. We noticed that the distributions tabulated in the memories were all Gaussian distributions having the same variance σ_ϕ^2 but with a different mean value μ_{ij}. Each OP block has its own mean μ_{ij} depending on the column (microphone and frequency) and on the line (grid position).

To overcome the memory issue, a natural way is to only store each mean μ_{ij} of each OP block (this memory block is called "ROM μ"). In this case, it is sufficient to tabulate the standard normal distribution computed with $\mu = 0$ and a specific σ_ϕ^2. This normal distribution is stored in a memory called *Rom Gaussian.* The probability value for each OP can be calculated by doing:

$$k - \mu$$

with k the sensor value and μ the corresponding mean value for the Gaussian distribution for the given column and line of the machine. This obtained value is taken as an address to the *Rom Gaussian* to obtain the final probability value. Instead of computing $k - \mu$ to calculate the address, the absolute value is taken $|k - \mu|$. Since the Normal distribution is symmetric around its mean value, by taking the absolute, we only need to store the positive

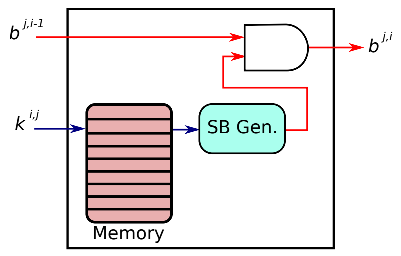


Figure 6.8: The OP block before introducing the on-chip normalization processing with the memory block containing an entry for each sensor value.

part of the Gaussian distribution centered in 0 and divide by 2 the needed registers. Finally, the value provided by the Gaussian distribution is stored in a register in the actual OP, as illustrated in figure 6.9.

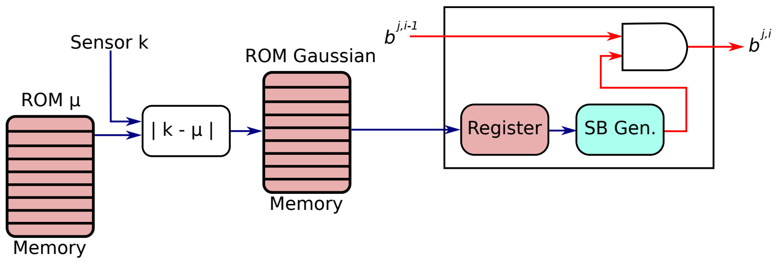


Figure 6.9: The mechanism used for the on-chip computation of the likelihood value for each OP block.

Naturally, this introduced the need to compute the probability value for each OP block prior to the computation which is done during an initialization phase. The initialization phase is triggered by the global control block whose behavior is described in figure 6.7. Due to the "filter"-like architecture of our VHDL circuit, the machine has to go through this initialization phase when switching to a new slice. Hence, the computation of the probability value can be parallelized. In order to achieve a reasonable trade-off between memory usage and duration of initialization phase, we decided to parallelize among all columns. Hence, we store p identical *Rom Gaussian* memories with p the number of columns. The detailed architecture can be seen on the picture 6.10.

In the past, when the likelihood values were pre-calculated off-line, the values were max-normalized as described in section 4.2.5 before being stored in the memory. As presented above, the probability values are now computed

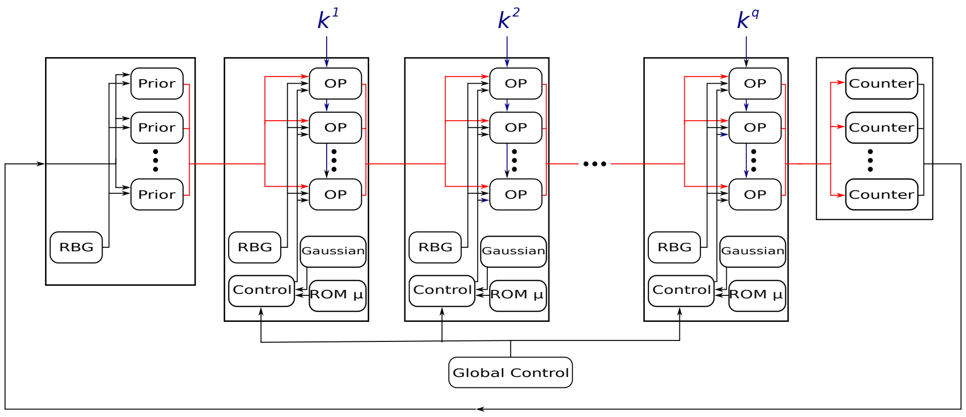


Figure 6.10: The detailed architecture of the VHDL implementation of the BM-sliced for the SSL.

on-line. Hence, the max-normalization needs to be performed on-chip as well before computing the inference. However, the max-normalization is very expensive in computation power when performing it on the hardware. Therefore, to avoid the additional computation power, we max-normalized the normal distribution stored in the *Rom Gaussian*. This is simply done by removing the normalization term in the definition of the normal distribution. The obtained distribution is max-normalized with its highest value being 1:

$$f(x|\mu,\sigma^2) = \cancel{\frac{1}{\sqrt{2\pi\sigma^2}}}\, e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Let us calculate how much memory is saved using this method. Looking at our standard setup, we will calculate the necessary memory in the past and the memory requirements using the developed on-chip method explained above.

Let *nb_bits* be the number of bits used to represent a probability value, *nb_word_tb* the number of values the sensor can have (this will give the number of lines in each memory block), *nb_line* the number of lines of the Bayesian machine (for the SSL task, this represents the cells in the localization grid), *nb_col* the number of columns in each slice and *nb_slice* the number of slices.

In the original version of the machine, without *Rom Gaussian*, we needed:

$$\begin{aligned}
&nb\_bits * nb\_word\_tb * nb\_line * nb\_col * nb\_slice \\
&= 8 * nb\_word\_tb * 64 * 10 * 10 \\
&= nb\_word\_tb * 512 * 100 \text{ bits} \\
&= nb\_word\_tb * 6,4 \text{ kByte}
\end{aligned} \tag{6.14}$$

Now, using the *Rom Gaussian*, we need:

$$\begin{aligned}
&nb\_bits\_mu * nb\_line * nb\_col * nb\_slice + nb\_bit\_G * nb\_word\_G * nb\_col \\
&= 10 * 64 * 10 * 10 + 8 * 1024 * 10 \\
&= 64 * 10^3 + 80 * 10^3 \\
&\approx 144 * 10^3 \approx 18 \text{ kByte}
\end{aligned} \tag{6.15}$$

with *nb_bits_mu* the number of bits used to represent the mean value μ of each OP block, *nb_bit_G* the number of bits used to represent each value stored in the *Rom Gaussian* memory and *nb_word_G* the number of values stored in the *Rom Gaussian*.

One can easily notice that the memory usage was $nb\_word\_tb{*}6,4$ kBytes without *Rom Gaussian* compared to roughly 18kBytes with the memory optimization using the *Rom Gaussian*. When the sensor is discretized on more than 3 possible values, the improved version of computation is cheaper in terms of memory. In practice, the number of possible sensor values is high and thus the memory usage increases accordingly.

However, as mentioned before, the on-chip computation of the probability values requires an initialization phase between each slice which needs some time during, which the BM can not start computation. In the current implementation, the initialization phase takes 192 computation steps in each block in parallel for each column. Since we have 64 lines in each column, we need $192/64 = 3$ steps per OP block. The 3 steps per line are:

1. looking up the mean value μ
2. calculating the $|k - \mu|$
3. look up the probability value in the dedicated *Rom Gaussian* memory.

Using pipelining techniques, we can speed up the initialization phase.

The pipelining principle is to parallelize the phase as much as possible instead of performing the computation sequentially, as showed in figure 6.11. Assuming the computation for the different lines is independent, we can parallelize the computation. Every line needs 3 computation steps which

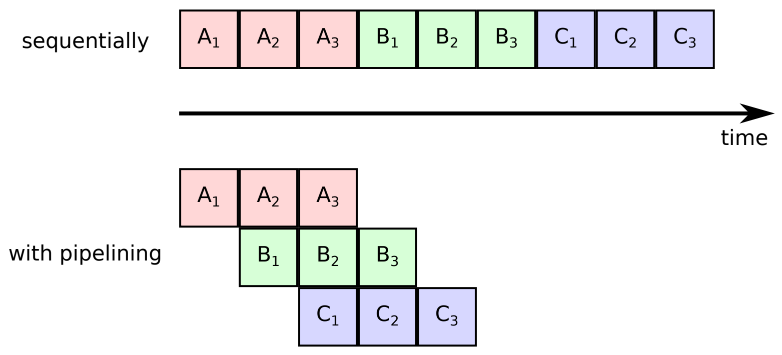


Figure 6.11: The principle of the pipelining technique used to speed up the initialization phase.

are done sequentially. Instead of waiting the end of the computation of the 3 steps for line A (as shown in the figure), we can start the computation for the line B once the first step of line A is done. Using this technique, we reduce the required steps to $nb\_line + (nb\_steps - 1) = 64 + (3 - 1) = 66$ steps with nb_steps the number of steps required for one line. In total for one slice, the machine needs the time needed to compute the inference and the steps needed for the initialization phase. However, the 66 steps for the initialization phase are negligible beside the computation time. Hence, the length of the initialization phase is linear to the number of lines in the machine. Note that in theory the initialization phase of a given slice can already be started while the previous slice is still processing. This would only require additional registers to store the computed likelihoods for the next slice.

6.5 Simulation environment

In this section we provide an overview of the different simulation tools used to validate the different parts of the systems.

6.5.1 Simulation data flow

Figure 6.12 aims to provide a schematic overview of the different possibilities to simulate the system. In blue all the blocks that are computed on a laptop. In green the parts that can be simulated on an FPGA. In red, the parts that can be run in real world.

To obtain the sounds recorded by the microphones, we can simulate the

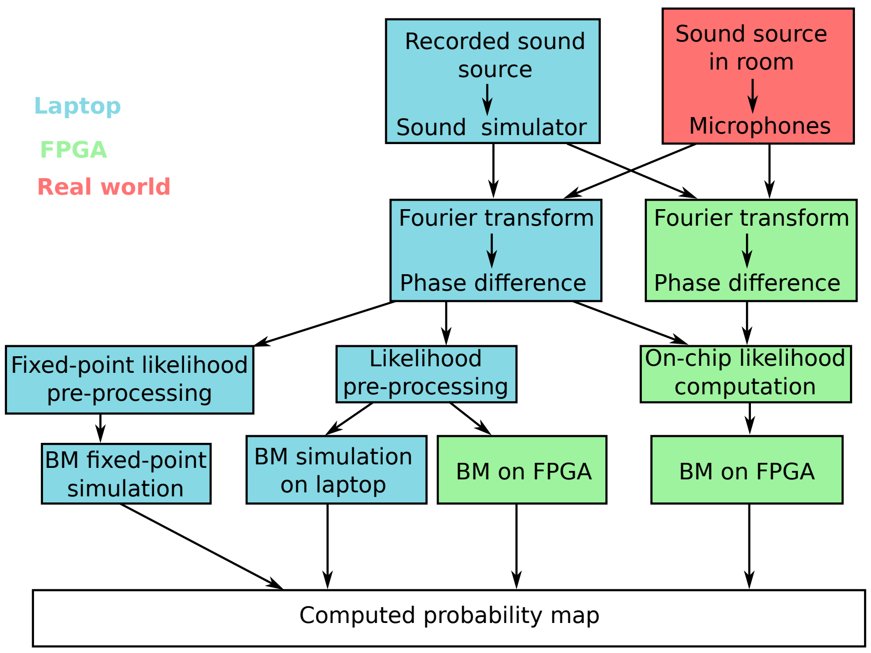


Figure 6.12: Simulation data flow showing the different simulation possibilities.

sound by the sound simulator as described in section 6.5.2 or record the sound using microphones in the experimental setup described in section 6.7.2. To compute the inter-channel phase differences (IPDs) needed to perform the inference, we can compute the Fourier transform and the IPDs on a laptop or on-chip as described in section 6.7.1.

The computation of the likelihoods can be performed by our simulator (described in section 6.5.3) in floating point or in fixed-point to be as close as the electrical circuit. Moreover, the likelihoods can be computed on chip as explained in section 6.4.2. Finally, to compute the inference on the probabilistic model, it can be computed on our BM simulator in fixed-point precision or in floating point. The Bayesian machine can also be run on FPGA to emulate the electrical circuit.

6.5.2 Sound pre-processing

Prior to the Bayesian Machine, which computes the sound localization, pre-processing is needed to generate the input for the BM.

Since for the experiments, we need to run the SSL for different source positions, we decided to generate artificial recordings. Moreover, we used a sound simulator to be able to easily test different acoustic setups. Therefore, we used the room impulse response (RIR) simulator of AudioLabs Erlangen [61]. As input for the RIR simulator, we used a mono-channel 16-kHz source speech signals which was recorded in an anechoic chamber. Given a specific room size, microphone positions and source position, the RIR simulator generates the recorded sound of the different microphones containing the reverberations.

Once the recordings for the different microphones are generated, the system preprocesses the signal in order to obtain the right input format required by the BM. Furthermore, for each microphone, the Fourier transform is computed in order to calculate the InterChannel Phase Difference (IPD). The IPD is saved into .mat files which can be imported by the simulation tools described in section 6.5.3 or converted into .mif files (Memory Instantiation File) to be run on the FPGA. The RIR simulator and the pre-processing are done in Matlab.

We also run experiments in real environment, which will be described in a dedicated section.

6.5.3 BM simulation tools

In order to validate the concept of the BM-sliced, we first coded it in a software simulation tool. A software simulator of the BM and the BM-sliced was developed using Python. The core of the computation was implemented in C++ using the Swig [2] interface to speed up the simulations. Using our simulator, we are able to get an early fast and deep insight of the internal behavior of the machine. Moreover, we can print the value of each variable present in the architecture and validate their correct values. This allows us to quickly test a set of parameters and find the best suited value for each parameter.

Furthermore, this step is important to test the principles of the algorithm and before validating the prototype in VHDL and run it on an FPGA, as presented in section 6.4.1.

A second version of the simulator was implemented to simulate the circuit at the bit level. Therefore, the LFSR and the fixed point operators were implemented in python/C++. This also allowed us to validate the circuit and be sure that the VHDL implementation of the circuit did not contained any bugs.

[2]Swig interface generator: `http://www.swig.org/`

Moreover, we can compare the different versions of the BM with and without slices and analyze for which configuration the sliced version of the machine accelerates the computation.

6.5.4 Standard experimental setup

To test our SSL system, we mainly used the same standard set of parameters which contains parameters for the audio part in Matlab for the sound generation and pre-processing but also parameters for the BM computing the SSL.

In the following, we will describe these parameters and split them into the two categories mentioned before.

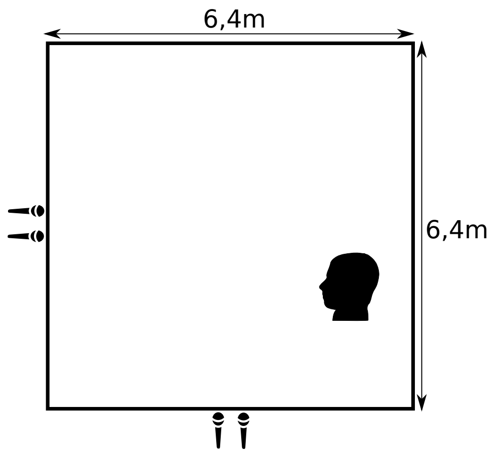


Figure 6.13: Setup of the room simulated to generate the sound for our experiments.

For the sound generation and the audio pre-processing, our standard case contains the parameters:

- Room size: 6.4m x 6.4m
- Source position: (1.2,4.4)
- Reverberation time: 150ms
- Sound: F1 (sound file used given to the RIR simulator)[3]
- Frame: 1 (number of the frame of the FFT used for the SSL)

[3]this sound file is recorded in an anechoic chamber without reverberations

- Reverberation order: -1 (maximum order of reverberations computed by the sound simulator)
- Microphone positions: (0,3.175), (0,3.225), (3.175,0) and (3.225,0)
- Inter-microphone distance: 5cm

Figure 6.13 shows the setup use for our standard case. The parameters for the BM for the standard case are:

- Grid size: 8 x 8 tiles = 64 lines in BM
- Number of slices: 10 slices
- Columns per slices: 10 columns
- Counter max: 8 bits [4]
- Bits for probability representation: 8 bits
- Shared random number generator: 1 LFSR per column

Note that the room size is 6.4m x 6.4m and the grid size is 8 x 8. This results in a localization in a grid whose tiles squares of 80cm side length. However, the grid size is not fixed. Depending on the experiments, the grid size was changed.

6.6 Simulated experiments

The goal of this section is to evaluate the sound source localization system described previously in simulation. We focus on a few experimentations on simulators that show the most interesting insights.

6.6.1 Localization performance

In this section we look at the localization performance of the developed system. In this particular case we took a 64 x 64 grid. Figure 6.14 shows the probability map obtained after running the BM-sliced for 5,000,000 steps. Typically, running the machine for such a large number of steps provides a probability map with high precision. The green point gives the position of the maximal counter, i.e. the line of the BM-sliced with the highest counter, representing the maximum of the probability distribution. The real

[4]this represents the Resampling Threshold (RT) between each slice

position of the source is given by the red point. Clearly the localization is not perfect, though the two points are globally in the same region. It is important to note that the localization obtained with the BM-sliced is similar to the one obtained with the SSL state-of-the-art EM (Expectation Maximization) algorithm of [40] using the same setup (shown as orange point in figure 6.14). It also is very close to the localization provided by exact inference (i.e. blue point which shows the floating-point calculation of (6.13)). Of course all methods were fed with the same sensor information). Therefore, the difference between estimated and true localization is not due to the accuracy of the inference of the BM-sliced but it is rather due to i) the low amount of used STFT frames, ii) the limited number of microphone pairs, and iii) the approximations made by the SSL model are not sufficient for this situation. In particular, room reverberations which are neglected in the model probably perturb the localization, since the speaker is located relatively close to the walls.

6.6.2 BM-standard vs. BM-sliced

One of the major improvements done on the Bayesian Machine was the introduction of the slices in the BM to obtain the BM-sliced, described in section 6.3.1. In this section, we aim to deeply analyze the impact of the slices on the computation and its impact on the temporal dilution. Since the distributions provided by Bayesian machines are approximate and the accuracy increases with computation time, it is interesting to inspect the results at different computation steps. Therefore, we will compare the computation speed and output accuracy of the BM-standard and the BM-sliced.

First, figure 6.15 displays the sum of all output counters of the machine as a function of the number of computation steps. This plot illustrates the temporal dilution since it shows how many "1"s are produced at the output of the machine. One can observe that both architectures have a certain "warm-up" time before providing a first significant approximated result. The BM-sliced bars (lightblue) raise considerably between 1,000 and 5,000 steps. This is due to the re-sampling threshold (RT) which is set to 128 for this experiment. Since we have 10 slices in the machine, we need to wait at least $10 \times 128 = 1{,}280$ steps before getting a first output. The standard BM architecture (red bars) exhibits a much longer warm-up time which is clearly due to the time dilution problem. Moreover, the sum of all output counters reaches a much higher value after 5,000,000 steps with the BM-sliced (about 1,728,000,000) than with the Standard-BM (only 31,741).

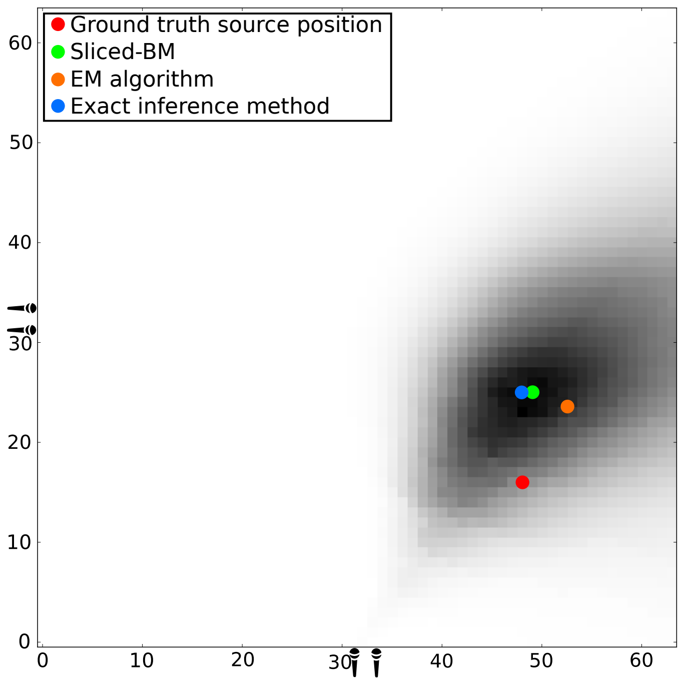


Figure 6.14: Probability map for BM-sliced at 5.000.000 steps. Maximum points of Ground truth source position (red), Source position estimated by the BM-sliced (green) which stays stable after 10.000 steps, Source position estimated by the EM algorithm of [40] (orange), Source position estimated by exact inference method (blue).

Now, we inspect the accuracy of the posterior probability distribution computed by the BMs, denoted P_{exp}. To this aim, we first calculate the theoretical probability distribution P_{th} corresponding to an exact inference method, directly combining (6.12) and (6.13). Then we calculate the Kullback-Leibler divergence (KLD) between P_{exp} and P_{th}:

$$\mathrm{KLD}(P_{th}, P_{exp}) = \sum_i P_{th}(i) \log \frac{P_{th}(i)}{P_{exp}(i)}. \tag{6.16}$$

The KLD is a classical measure of the "distance” between two probability distributions.[5] figure 6.16 displays the KLD values for both Standard-BM and BM-sliced, as a function of the number of computing steps. Note that,

[5]Although not symmetric, the KLD behaves as a distance. In particular it is positive and a KLD equal to 0 indicates that the two distributions are identical.

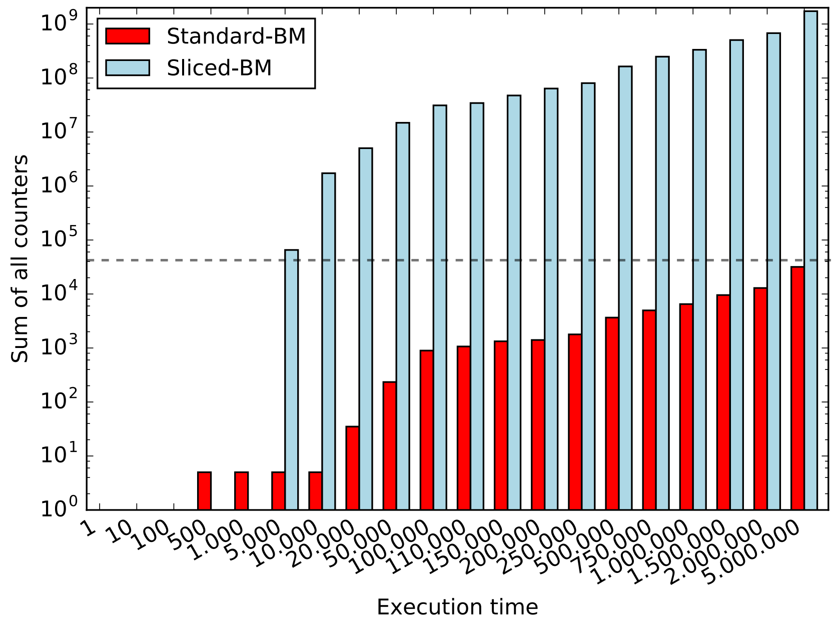


Figure 6.15: Sum of all output counters (on a log-scale) as a function of the number of computation steps.

since each BM needs some "warm-up” time as shown in figure 6.15, the plotted bars start at the number of steps where the machine computed enough useful information for comparison with the exact inference. The KLD value difference between Standard-BM (in red) and BM-sliced (in lightblue) is due to the re-sampling method which strongly reduces the temporal dilution. In the Standard-BM, the number of "1s" present at the end of all columns (and incrementing the final counters) is very low. However, in the BM-sliced, the number of "1s" is much higher so is the number of incremented counters at the output level. This allows to obtain a much faster and a much better approximation of the target distribution. For example, after only 5,000 steps the BM-sliced nearly gets the same KLD value as the Standard-BM after 5,000,000 steps, hence an acceleration factor of 10^3.

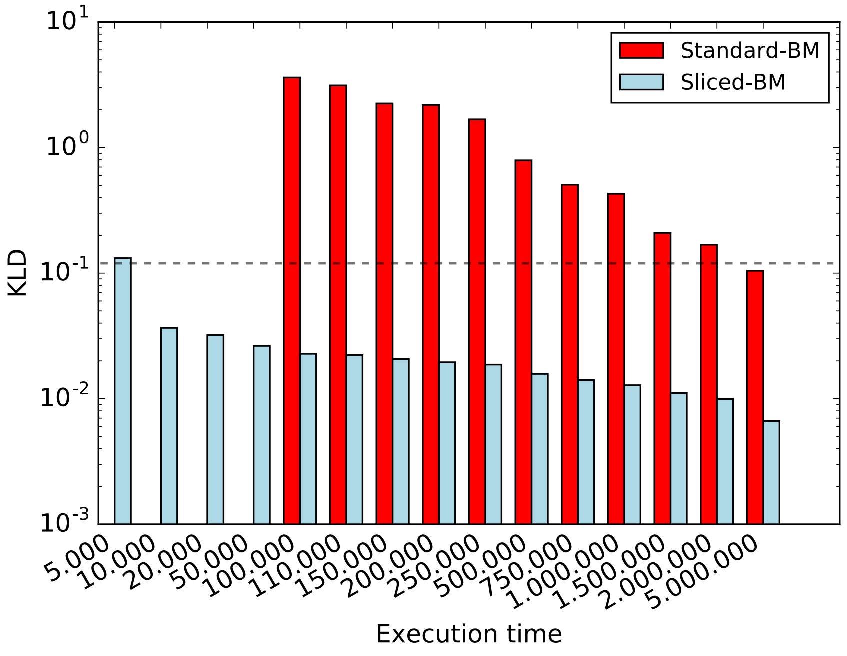


Figure 6.16: Comparison of the Kullback-Leibler divergence between the distribution computed by the BM (standard and sliced version) and the exact distribution as a function of the number of computation steps.

6.6.3 Impact of the probability discretization

This section will analyze how the machine behaves when varying the number of bits used to represent the probability values, namely *n_bit*. Moreover, the impact on the quality of the probability distribution is measured using the KLD. Notice, that the *n_bit* parameter directly impacts the size of the corresponding circuit of the machine and hence its power consumption. It also defines the width of the memory blocks of the BM since the likelihoods $P(K_i|S)$ are stored in registers in each OP block of the machine. Furthermore, we analyze how the KLD evolves for different values for the re-sampling threshold (RT). The RT is the maximum value reached by at least one counters (one line of the machine) before moving to the next slice. Since we target a low-power architecture, it is crucial to find the optimal trade-off between the resulting precision and the circuit activity to minimize consumption.

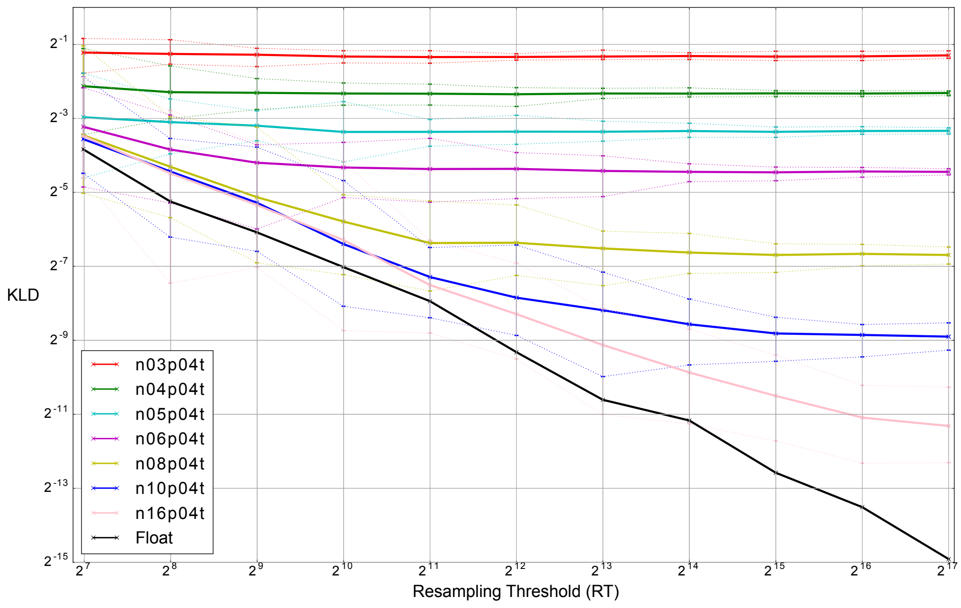


Figure 6.17: KLD as a function of the re-sampling threshold value for different *n_bit*. *n00p04t* being the exact inference.

Figure 6.17 shows the results of the experiments. The plot shows the KLD value (Y-axis) as a function of the re-sampling threshold value (X-axis) for the different values of *n_bit* (different lines). The sensor discretization was set to 4 bits in all cases (*p04*) which means that there were 16 different possible values for the measured phased difference. A linear feedback shift register (LFSR) was used as a random number generator (RNG) which leads to the extension of the name *p04t* ('t' for use_LFSR = True). For each value of the probability discretization *n_bit* (*n*03 to *n*16) 20 runs of the same experiment were made to calculate a mean value, which is given by the thick line. The minimal and maximal KLD of each the 20 runs is provided by the dashed lines respectively underneath and above the line of the mean value (thick line). The black line (n00p04t) represents the float computation of the probability distribution as a reference of a maximal precision. Notice the logarithmic scale in both dimensions for a better presentation of the obtained data. The experiments were made for re-sampling thresholds varying from $2^7 = 128$ up to $2^{17} = 131072$.

Looking at the mean lines (thick lines), one can clearly observe a convergence to a maximal precision which depends on the value of *n_bit*. The precision of the distribution improves with *n_bit*. Moreover, in all cases, the increment of the re-sampling threshold (RT) leads to a better KLD. This is due to the fact that with a higher RT, the discretization of the probability value increases and hence the distribution can be represented with more detail. However, an high RT means a longer execution time for the inference since each slice of the machine will take more time to reach its RT. Therefore, it is crucial to adapt the RT to the given *n_bit* to avoid useless computation and consequently reduce the activity of the circuit.

6.6.4 Impact of the sensor precision

This section analyzes the impact on the computation when the sensor precision changes. Depending on the application that is treated by the BM, sensors may have different precisions. Therefore, it is important to analyze the overall result for different discretization of the sensor data.

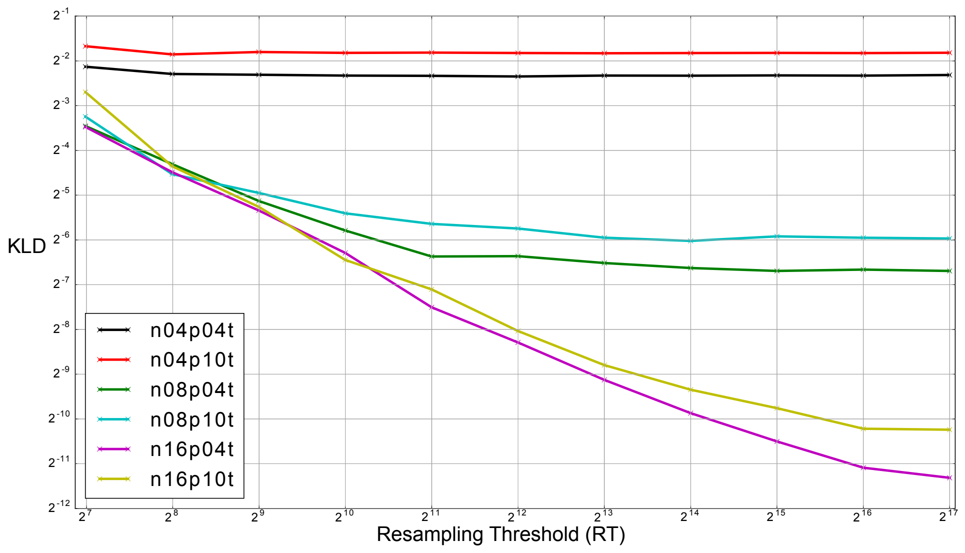


Figure 6.18: KLD as a function of the re-sampling threshold value for different discretization of the sensor data.

Figure 6.18 provides the results for different sensor precisions *p04* and *p10* which correspond respectively to 4 and 10 bits of precision for the sensor data. As the plot shows, the sensor precision does not massively impact the computation as the KLD value for each value of the RT does not vary much between *p04* and *p10* for a given value of *n_bit* (remember that both axes are in a logarithmic scale). Notice that for the clarity of the plot, we removed the plots for *p06* and *p08* since they were partially or completely overlapping with the lines for *p10*. As a conclusion, one can say that the sensor precision does not have a strong influence on the computation result whereas the choice of the *n_bit* heavily impacts the final result. This is not as astonishing as it might seem at first sight: given the large number of sensors in our application (the BM has a total of 100 columns), the exact product in (6.13) has a precision of 400 bits for 4-bit sensors, much more than can effectively be computed. Our computation is necessarily affected by rounding, and going to higher sensor precision cannot improve this situation.

6.6.5 Impact of the LFSR

Since, as explained in section 4.2.3, in the current implementation of the circuit, linear feedback shift registers (LFSR) are used as random number generators (RNG) to generate the stochastic bit streams. Therefore, we want to study the impact of the LFSR on the computed probability distribution. A succession of runs were made for fixed (*n02p08*, *n06p08* and *n08p08*) parameters of *n_bit* and sensor precision *p*. The computation is compared when using our standard RNG which is a 32 bit LFSR in the BM (i.e. lines with *t* at the end) as opposed to when the BM uses the RNG included in C++ (i.e. lines with *f* at the end). C++ uses Mersenne Twister as its internal RNG. Like in the previous sections, 20 runs were made for each value of RT (X-axis) and the mean value of all runs is plotted.

Figure 6.19 shows the KLD value as a function of the different values for the re-sampling threshold (RT). The precision of the computed distribution improves with increasing RT. However, one can see a notable difference in the KLD between the purple line (*n08p08f*) and the yellow line (*n08p08t*). At the RT value 2^{16}, a factor 16 between both lines can be observed which is clearly not negligible.

Therefore, as future work, we have to explore other and better ways to generate random bits while keeping the power consumption low. RNGs can sometimes require a large area on the circuit. For this reason, exploiting physical phenomena to build RNGs might be an important task in future for us.

6.6.6 Computation accuracy depending on the number of slices

One major parameter which was introduced with the Sliced-BM architecture is the number of slices contained in the machine. As the number of slices impacts the number of columns in each slice, this parameter determines the

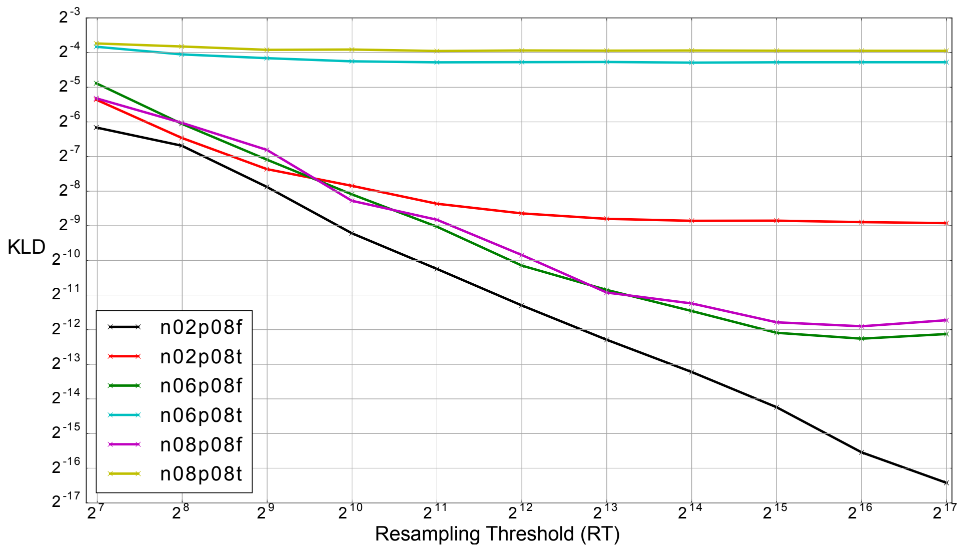


Figure 6.19: KLD as a function of the re-sampling threshold value for runs with LFSR as random number generators (t) and with the random number generator of C++ (f).

circuit size of the machine. Moreover, in terms of computation speed, the number of slices is a key parameter since a low number of slices means that re-sampling is performed only a few times and hence the risk of loosing time due to temporal dilution grows. On the opposite, when having many slices in the machine, the inference will take more time because the re-sampling will occupy a large part of the total computation time. The goal is to find a trade-off between the minimal circuit size and a reasonable computation time.

Figure 6.20 shows the evolution of the precision for different numbers of columns per slice in the machine considering the machine has 100 columns in total. The graph shows that the quality of the results improves with the number of columns per slice. However, if the slices become too large, the computation time of the machine can significantly increase due to temporal dilution. Hence, the need to find a reasonable trade-off.

To summarize the results presented in the previous sections, it has been shown that the discretization of sensor data has a lower influence on the computation. Furthermore, the discretization of the probability values in the machine, which has an important impact on the quality of results and impacts

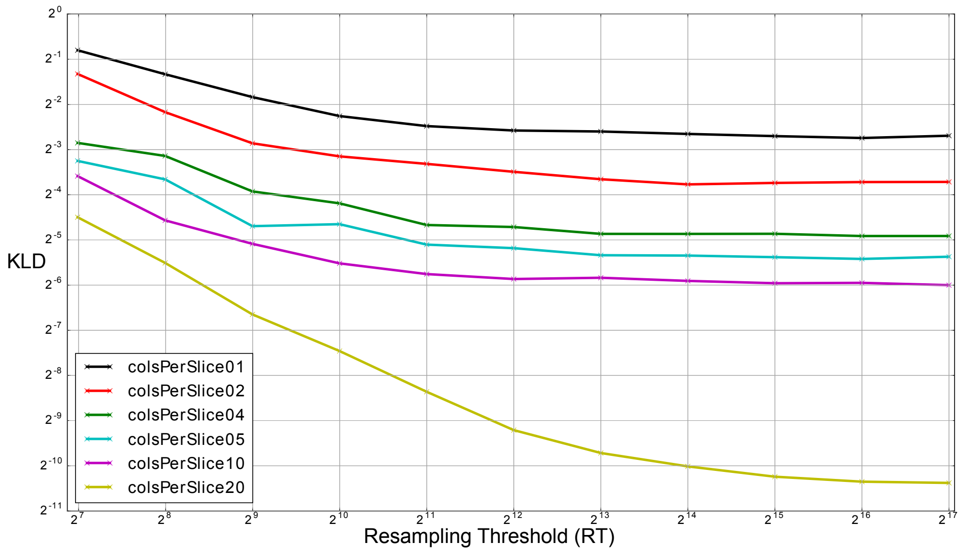


Figure 6.20: KLD as a function of the re-sampling threshold value for different number of columns per slice.

the activity. This parameter directly defines the circuit size. Moreover, it has been shown that the higher the re-sampling threshold (RT), the better the distribution computed by the machine. Furthermore, the impact of the quality of the random number generator (RNG) has been illustrated. Lastly, the evolution of the accuracy of the computation when the number of slices changes has been observed.

6.6.7 Power consumption measurements

Since we are working on new computing units and suggesting alternatives to conventional processors, we do a lot of basic experimentations to compare the developed architectures to the existing ones. Therefore, this section aims to analyze the power consumption of the proposed Bayesian machine. The easiest way to have the most realistic consumption measurements is to run ASIC (Application-specific integrated circuit) simulations. We performed ASIC simulations to obtain more realistic numbers about power consumption, circuit size and the maximal frequency we can run the circuit.

Since our architecture needs a lot of memory, we wanted to measure the impact of our optimization. For that reason, we compared both architectures, the original and the optimized one using the on-chip likelihood computation explained in section 6.4.2.

The design flow has been set up to synthesize the VHDL description of both circuits. The ASIC simulations were made using the FDSOI 28nm technology from STMicroelectronics.

The setup used for these measurements was close to the standard experimentation setup described before in section 6.5.4:

- 2 microphone pairs
- $50 \times 2 = 100$ columns in the Bayesian machine
- 10 columns per slice
- Grid: $8 \times 8 = 64$ tiles $\Rightarrow$ 64 lines per column

In terms of the architecture, the BM had:

- 64×10 OP blocks with each a registers of 8 bits to store the likelihood and an AND gate
- 11 LFSR as random number generators (1 per column + 1 for the prior column)
- 10 control blocks, each having 1 Finite State machine (FSM)
- Counters: 64 registers of 8 bits at the end of the 10 columns (at the end of each slice)
- Priors:
  - 64 registers of 8 bits
  - 64 AND gates

The memory was simulated by the Static Random Access Memory (SRAM) memory compiler of STMicroelectronics. Let us compare the memory needs in both architectures. In the original architecture (without the on-chip likelihood computation), each column (10 in total) requires

$$\begin{aligned}
&nb\_bits * nb\_word\_tb * nb\_line * nb\_slice \\
&= 8 * nb\_word\_tb * 64 * 10 \\
&= nb\_word\_tb * 5120 \text{ bits} \\
&= nb\_word\_tb * 640 \text{ Byte}
\end{aligned} \tag{6.17}$$

with *nb_bits* the number of bits used to represent a probability value, *nb_word_tb* the number of values the sensor can have, *nb_line* the number of lines of the Bayesian machine (for the SSL task, this represents the cells in the localization grid), *nb_col* the number of columns in each slice and *nb_slice* the number of slices. Typically *nb_word_tb* is set to 32 which leads to a memory need of 160 kByte per column.

The optimized machine needs a reduced amount of memory. To allow the on-chip likelihood computation, the machine needs 2 different memories per column, namely the *Rom MU* and the *Rom Gaussian*.

Each of the 10 *Rom MU* memory block has the following size:

$$\begin{aligned}
&nb\_bits\_mu * nb\_line * nb\_slice \\
&= 10 * 64 * 10 \\
&= 64 * 10^2 \text{ bits} \\
&= 0.8 \text{ kByte}
\end{aligned} \tag{6.18}$$

with *nb_bits_mu* the number of bits used to represent the mean value μ of each OP block.

The size of each of the 10 *Rom Gaussian* memory blocks is:

$$\begin{aligned}
&nb\_bit\_G * nb\_word\_G \\
&= 8 * 1024 \\
&\approx 8 \text{ kByte}
\end{aligned} \tag{6.19}$$

with *nb_bit_G* the number of bits used to represent each value stored in the *Rom Gaussian* memory and *nb_word_G* the number of values stored in the *Rom Gaussian*.

Circuit	Original	Optimized
Area (μm^2)	184,978	132,510
Max. Frequency (MHz)	1111	1136
Static consumption (μW)	356.72	149.61
Dynamic consumption (mW)	8.76	5.62

Table 6.1: Results of the ASIC simulations for both architectures: the original and the optimized version.

The results are given in table 6.1. The obtained numbers show that the optimized version of the circuit reduces the circuit area by approximately 30% and the power consumption by approximately 35%. This impressive reduction in area and consumption is achieve because memory represents a significant part of the circuit. In the optimized version, nearly 60% of the area is dedicated to memory and more than 55% of the power consumption is absorbed by the memory.

A remaining question is how the existing memory technology that we used in this experimentation could be replaced. As mentioned above, we used SRAM in the current set up. However, exploring alternatives such as Magnetic Random Access Memory (MRAM) form a promising replacement since the memory area could be replaced by a factor of 20 [24] and hence reduce significantly the power consumption. Note that both architectures have not been optimized for the simulation in terms of place and route and all the different optimization strategies that can be done to reduce power consumption.

6.7 Real world experiments

All experiments presented above were performed using sound generated by the (RIR) simulator of AudioLabs Erlangen [61]. Moreover, a not negligible amount of pre-processing was done on the laptop before feeding the FPGA with the appropriate information.

However, in order to get closer to the real world with our localization method, we worked on the realization of an autonomous demonstrator.

This work was mainly done by Jérémy Belot during his Master 2 internship during which I co-supervised him. Later he joined the team as an engineer to finalize the real world set up and perform additional experiments.

6.7.1 Pre-processing on chip

Until now, the pre-processing was performed by a computer before giving the phase difference information to the FPGA in order to make the adequate inference. In this section, all the modules that were added to the system are described.

Figure 6.21 provides a good overview of the implemented system. We mainly focused on the BM1 block which is the Bayesian machine and thus the core of the system. However, the pre-processing in red and the microphones in blue were added to do the experiments in real environment. A central control block has been added for synchronizing the sound acquisition, the pre-processing, and the inference.

One major step in this work was to implement the Fourier transform on the FPGA. We used the Cordic algorithm which computes the Fourier transform and computes the phase of the signal at the same time. Using this method allows to implement the FFT on a very small circuit area and thus

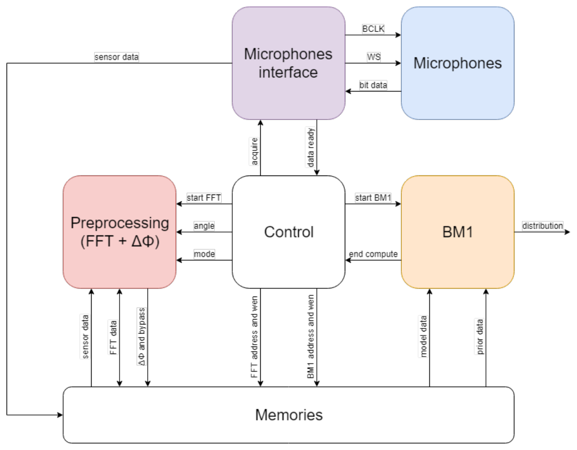


Figure 6.21: Overview of the different modules present in the global system and their connections.

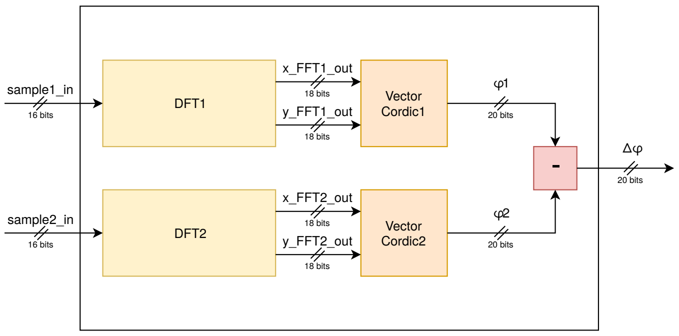
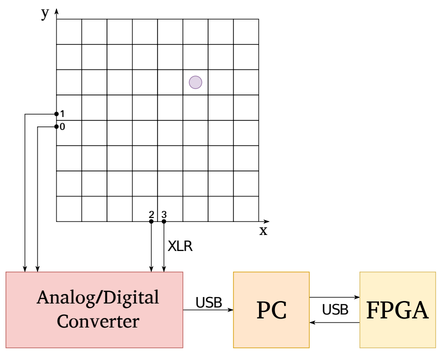


Figure 6.22: Phase difference computation module for 1 microphone pair.

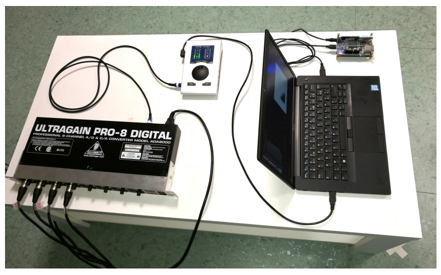
be cheap to produce [12]. As shown in figure 6.22, the FFT using the Cordic algorithm is done in parallel for 2 microphones of the same microphone pair before computing the Inter-channel Phase Difference (IPD).

6.7.2 Experimental set up

To test the system, we installed an experimental set up in a 3m x 3m room in which we put a 8x8 grid for the localization which leads to squared grid cells of 37.5cm side. Two pairs of microphones have been placed in the room as shown in figure 6.23 (a). In figure 6.23 (b), one can see the cables of the 4 microphones getting into the Analog/Digital Converter (at the bottom left). The digital information go to the external sound card (in white) via an optical connector. More precisely, we used a RME Babyface Pro as sound card. It is connected via USB to the laptop which loads the sound without any transformation into MIF files (Memory Instantiation File) on the FPGA to run the Bayesian machine. Note that the laptop is only used as a control unit and does not perform any sound pre-processing.



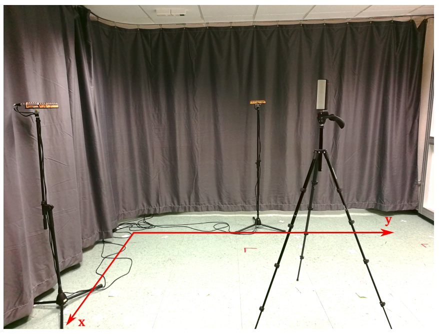
(a) Room



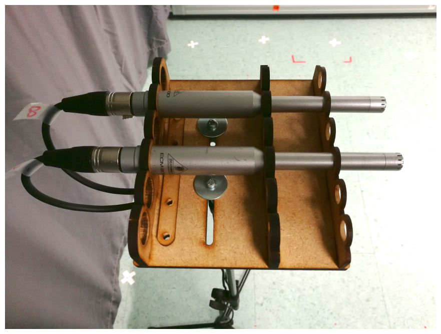
(b) Microphones

Figure 6.23: Sound acquisition set up.

Figure 6.24 shows how the microphones are mounted (a) and placed in the room (b). Note that 2 walls of the room are simulated by a thick curtain (in gray). The localization is performed in a plane at approximately 1.5m height. The source is simulated by a UE BOOM 2 bluetooth speaker. Thanks to the special mount for the microphones, the inter-microphone distance can be easily changed if needed.



(a) Room



(b) Microphones

Figure 6.24: Microphone set up in the room.

6.7.3 Results

The localization results on a single frame is provided in figure 6.25. In red the real sound position. In green the center of gravity of the distribution. In blue the cell with the maximum counter in the distribution. Note that the system is performing well for source positions close to the center of the room. However, when moving closer to the walls and the corners, the systems performance decreases, as shown in the figure.

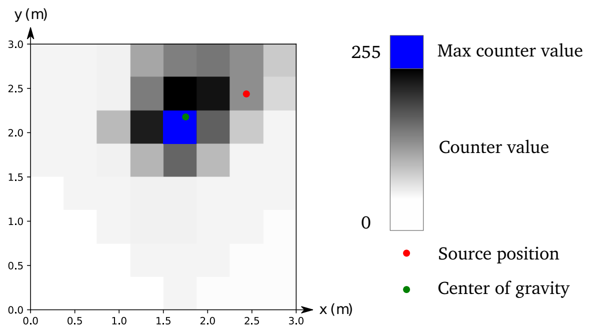


Figure 6.25: Localization map obtained using the free field model.

Moreover, it is important to consider the size of the time-frames used for the localization, which is very small.

6.7.4 Optimization - filtering

Considering the big changes in the performance of the system from one frame to the other, a filtering method was added to the system. The result of one frame is used as a prior to start the computation of the next frame.

Using this filtering method, the result quality considerably increases after only a few frames. As shown in figure 6.26, the result is already acceptable after frame 3 and becomes perfect at frame 4.

However, some frames are using a signal portion without a lot energy in the signal. The current system does not analyze the obtained signal by the microphones and simply runs the localization method.

6.7.5 Optimization - bypass

Adding a threshold to use signal portion which is useful for the system was the aim of this optimization. The module responsible for the Fourier transform also provides the Fourier coefficient which provides the information of the amount of energy present in the signal for a give frequency. Using this

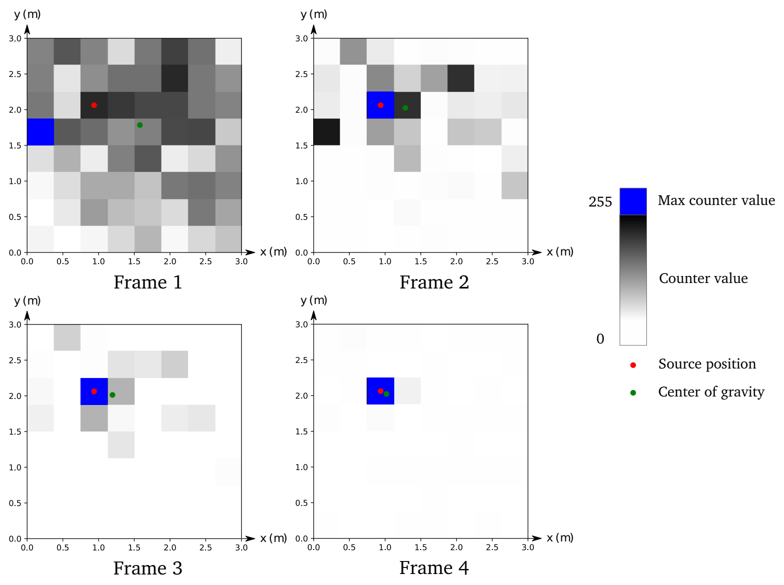


Figure 6.26: Evolution of the localization map after filtering over 4 frames.

information, we introduced a bypass signal which bypasses a specific column of the machine when the signal at the specific frequency is underneath a certain threshold.

Figure 6.27 shows how the system performs for different threshold values. Figures (a) and (b) show the results when using no threshold (NT) and filtering (F) or no filtering (NF). Figures (c) and (d) shows the performance when using a medium threshold (MT). When using the medium threshold, approximately 50% of the columns are bypassed. However, the system performs the best when using a high threshold (HT) where only 10 out of the 100 columns remain used for the sensor fusion. One can see that using filtering and a high threshold allows the system to perform perfectly even in the corners of the room, i.e. figure (f).

6.7.6 Optimization - learned model

A third optimization was done to test another probabilistic model on the machine. This approach is based on learning. Moreover, the theoretical Inter-channel Phase Differences (IPD) are learned to improve the localization performance. Therefore, on every possible position on the grid, a sound is emitted to learn the IPD for all the frequencies at this location. Due to the learning, the theoretical IPDs used in the model are much better quality and hence improve the system's performance. We trained by using a sound which consists of the sum of all the sinus signals at the frequencies which we use to localize the sources.

The results of the localization drastically improved due to this approach as illustrated in figure 6.28.

In order to provide an overview of the performance of this method, the results for a few source positions are given in figure 6.29, figure 6.30, figure 6.31, figure 6.32 and figure 6.33.

Even though the performance is remarkable, one have to keep in mind how restrictive the learning process is since when installing the system in a room, the learning process has to be done every time. Also, one should emphases how bad this method scales when the grid size increases.

6.7.7 Circuit area

Beside the localization performance of the system, we are also interested by having a system which is as small as possible in terms of circuit area. Therefore, it is important to look in the area of the different modules of the machine. When mapping the circuit on an FPGA, we can easily obtain the area of each part of the system by looking at the number of Adaptive Logic Modules (ALMs) needed by each module.

Hierarchy	Resources FPGA (Comb ALUTs)	Memory RAM (bits)
top_SSL_FPGA	15039	381605
→ top_SSL	13923 (100%)	315432
→ top_preprocess	4927 (35%)	131112
→ top_BM1	8931 (64%)	184320
→ BM1_core	7030 (50%)	0
→ Control	92 (0.7%)	0
→ Memory_module	1803 (13%)	184320

Table 6.2: Circuit area measurement of the system mapped on the FPGA.

In table 6.2, one can see that the Bayesian machine does only occupy 66% of the system. Unfortunately, 32% is needed for the preprocessing. Due to the Fourier transform, this module is very large.

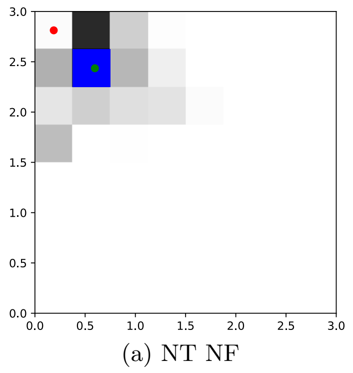
6.8 Conclusion

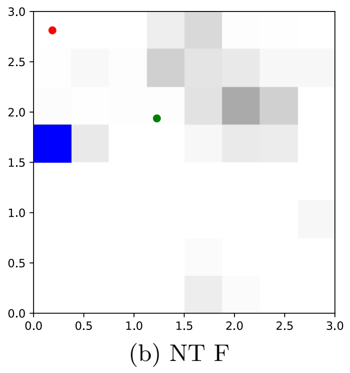
In this chapter, a probabilistic method to perform sound source localization has been presented. This method is working in the time-frequency domain.

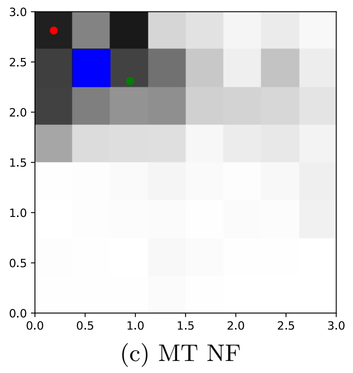
The probabilistic model has been introduced as well as the adjustments on the Bayesian machine to run the inference on this Bayesian model. Moreover, an optimization of the machine has been described, which speeds up the computation by a factor of up to 10^3. Also, to run the machine on an FPGA, the architecture has been implemented in VHDL and an on-chip likelihood computation method has been presented in order to reduce the memory usage of the machine. Further more, several simulations have been performed to analyze the impact of the different parameters of the machine and study the power consumption of this hardware. Finally, the localization system has been confronted with the real world using a test environment with real microphones placed in a room.

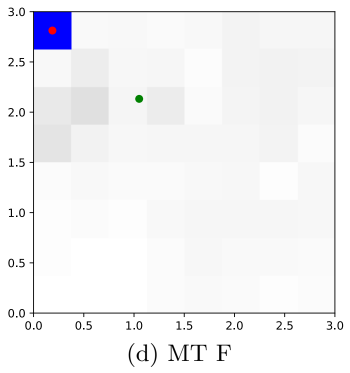
Moreover, to increase the performance of the system, one could have used some additional pre-processing such as dereverbation algorithms to compute the measured phase difference (IPD). However as the performance of the system is already very good, we do not aim to achieve state of the art results in the localization. The main goal of our project was more focused on proposing alternative computing architectures and use them to deal with common tasks.

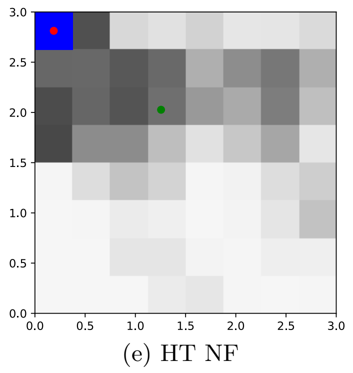
As this method is working in the time-frequency domain, it requires to compute the Fourier transform for the recorded signals. Analyzing the VHDL circuit for the SSL application, we noticed the significant area dedicated to the Fourier transform. Therefore, in the next chapter we present another sound source localization method which works in the temporal domain and hence does not require anymore the computation of the Fourier transform.


(a) NT NF


(b) NT F


(c) MT NF


(d) MT F


(e) HT NF

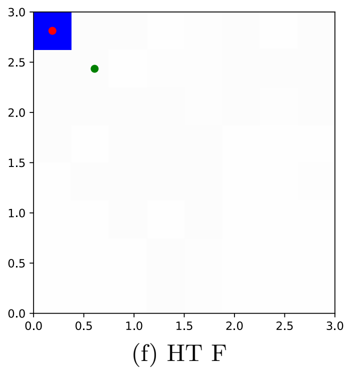

(f) HT F

Figure 6.27: Examples of localization when using no threshold (NT), a medium threshold (MT) and a high threshold (HT) using the filtering method (F) or without filtering (no filtering - NT).

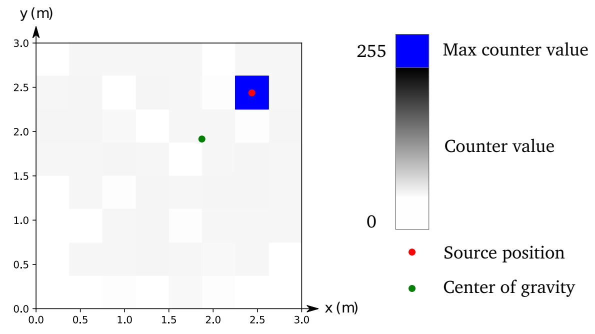


Figure 6.28: Localization map using the learning method at position 54.

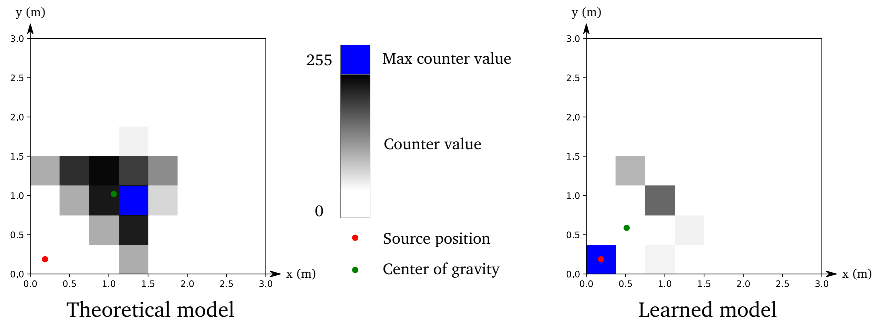


Figure 6.29: Comparison between the original free field model and the learned model at position 0.

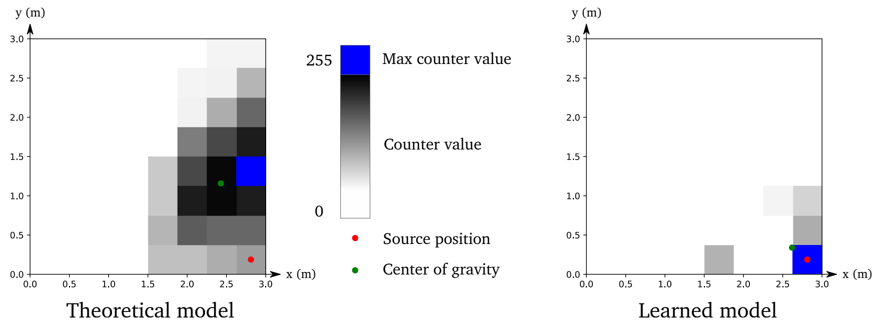


Figure 6.30: Comparison between the original free field model and the learned model at position 7.

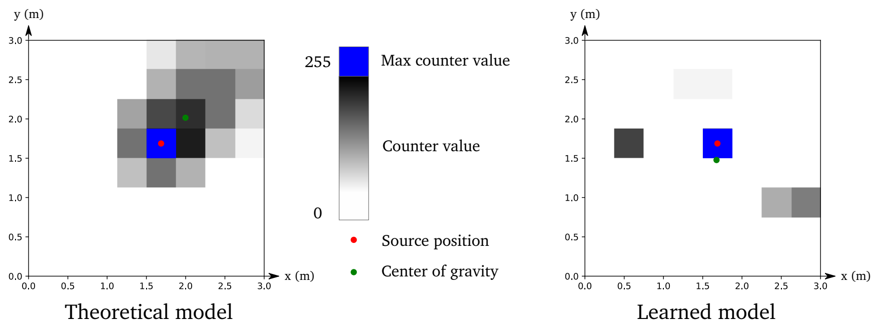


Figure 6.31: Comparison between the original free field model and the learned model at position 36.

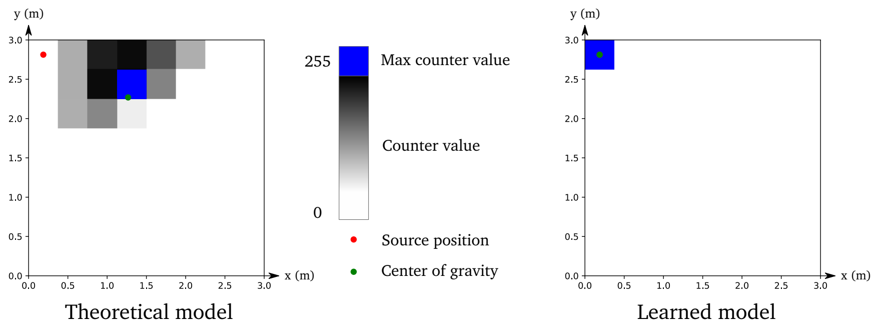


Figure 6.32: Comparison between the original free field model and the learned model at position 56.

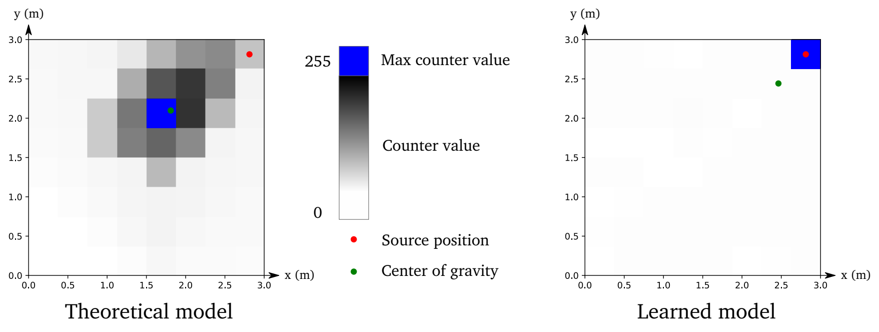


Figure 6.33: Comparison between the original free field model and the learned model at position 63.

Chapter 7

Mono-Source localization in the temporal domain

In the previous chapter, a solution for sound source localization (SSL) has been presented. This work was based on using the free field hypothesis and the Inter-channel Phase Difference (IPD) as features for the localization. However, the computation of this features require the Fourier transform of the signals recorded by the microphones. Since nowadays most of the source localization methods are working in the time-frequency domain, we decided to work on that concept.

However, as shown at the end of the previous chapter, we analyzed the area of the circuit of the different modules in the final system and noticed the important area of the module responsible for the Fourier transform. This component is taking approximately 35% of the entire circuit area. As our research project aims to design more lightweight solutions, we worked on algorithms for sound source localization without using the Fourier transform.

The two methods presented in this chapter works entirely in the temporal domain. The basic idea is to compare the temporal signal recorded by the different microphones and extract the location of the source. One method uses the attenuation to compute the source position whereas the second method does not but assumes several pairs of nearby microphones.

7.1 Probabilistic model with attenuation

For this model we consider the following stochastic variables will be used:

- $\boldsymbol{M}$: the matrix of the stochastic variables associated to the sounds recorded by the microphones with M_t^i the stochastic variable associated to microphone i at time t and m_t^i its value.

- $\boldsymbol{C} = (C_x, C_y)$: the probabilistic variables associated to coordinates of the sound source with $C_x \in [0, \dim_y]$ and $C_y \in [0, \dim_y]$ with $\dim_x$ and $\dim_y$ the dimensions of the room in the x and y axis. The cardinality of this probabilistic variable is $w = card(C_x) \times card(C_y)$.

Let be T the total recording time expressed in number of samples, I the number of microphones used. The values of the stochastic variables associated to the sounds recorded by the microphones are discretized between $-m_{\max}$ and $m_{\max}$ and so, the values of the m_t^i are taken in this discrete set : $m_t^i \in \{-m_{\max}, \ldots - 1, 0, 1, \ldots, m_{\max}\}$.

The localization is made in small time intervals called frames. A frame is a set of T consecutive samples taken during this time window. It is assumed that during each frame the sound source does not move.

The basic idea is to analyze the offset between one selected microphone and the other microphones. For the sake of simplicity, let microphone 0 be the reference microphone. For each position on the grid, one can compute the offsets between the microphone 0 and the other microphones.

First, before going into details, let us define two functions needed in the model. Given the source location $c = (c_x, c_y)$, and the position of the reference microphone 0 and the microphone i, one can compute the "ideal" signal TDOA (time difference of arrival) in between the two microphones expressed in number of samples. Physically, $\tau_{i_1}^{i_2}(c)$ gives you the number of samples for the microphone i_2 and microphone i_1 to receive the same signal:

$$\tau_{i_1}^{i_2}(c) = (d(c, i_2) - d(c, i_1)) \cdot \frac{F_s}{v} \tag{7.1}$$

where $d(c, i)$ is the distance between the source and the microphone i, v is the speed of sound ($\simeq 340$ m.s^{-1} in the air) and F_s is the sampling frequency. Note that $\tau^i(c) = \tau_0^i(c)$ is the short form that gives you the number of samples for the microphone i and microphone 0 to receive the same signal.

Moreover, we need the function $a(c, i)$ which provides the attenuation factor for a given distance. It follows the attenuation model which can be found in Equation (1) in [52]:

$$a(c, i) = \frac{1}{\sqrt{4\pi} \cdot d(c, i)} \tag{7.2}$$

Having specified the probabilistic variables and the helping functions, we can defined the decomposition of the probabilistic model. Knowing the localization of the source and the recording on the reference microphone, the stochastic variables attached to the other microphones are independent due to the free field hypothesis and the joint distribution on $\boldsymbol{C}, \boldsymbol{M}$ is written as follows:

$$P(\boldsymbol{C}, \boldsymbol{M}) = P(\boldsymbol{C})P(\boldsymbol{M}^0|\boldsymbol{C}) \prod_{i=1}^{I} \prod_{t=0}^{T} P(\boldsymbol{M}^i|\boldsymbol{C}, \boldsymbol{M}^0) \tag{7.3}$$

As we do not have any prior knowledge on the localization of the source, we model $P(\boldsymbol{C})$ as the uniform distribution. Moreover, $P(\boldsymbol{M}^0|\boldsymbol{C})$ is not needed since $\boldsymbol{M}^0$ is in the evidences. Given a sound source emitting from position c, $M^i_{t-\tau^i(c)}$ only depends on M^0_t. It follows a Gaussian distribution centered on $\frac{a(c,i)}{a(c,0)} \cdot m^0_t$ with a variance σ^2, where σ^2 represents the precision of the microphone:

$$\boldsymbol{M}^i_{t-\tau^i(c)} \sim \mathcal{N}(\frac{a(c,i)}{a(c,0)} \cdot m^0_t, \sigma^2) \tag{7.4}$$

In practice, our probabilistic model for SSL consists of a series of distribution values

$$P(m^i_{(t-\tau^i(c))}|\boldsymbol{m}^0 c) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(- \frac{(m^i_{(t-\tau^i(c))} - \frac{a(c,i)}{a(c,0)} \cdot m^0_t)^2}{2\sigma^2} \Big) \tag{7.5}$$

which are evaluated for

1. each point of the 64×64 source position grid
2. a series of time samples t.

Given the signal recorded by the microphones m, we can use the likelihoods $P(m^i_{(t-\tau^i(c))}|\boldsymbol{m}^0 c)$ from equation 7.5 to compute the posterior probability distribution over $\boldsymbol{C}$ using:

$$P(\boldsymbol{C}|\boldsymbol{m}) \propto \prod_{t=0}^{T} \prod_{i=1}^{I} P(m^i_{(t-\tau^i(c))}|\boldsymbol{m}^0_t, \boldsymbol{C}) \tag{7.6}$$

The goal is to run the inference on the Bayesian Machine (BM). Therefore, the machine has been adapted to evaluate the expression above, which will be described in the next section.

7.1.1 Bayesian machine adapted to SSL

The architecture used for the inference is the Bayesian machine which was also used in the SSL with the Fourier transform. However, the inputs changed since now the signal recorded by the microphones was directly provided to the machine to compute the likelihoods and perform the inference.

In practice, we used 4 microphones in total where m^0 was used as the reference microphone. This means that for each time sample t, the signal of the reference microphone is compared to the 3 others microphones using the likelihoods of 7.5. Moreover, as we work on frames of T samples, we obtain $3 \times T$ likelihoods to multiply and have $3 \times T$ columns in the BM.

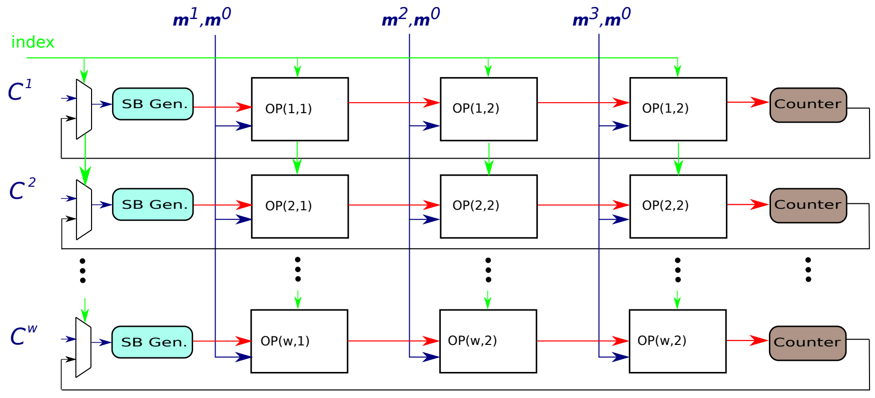


Figure 7.1: Architecture of the Bayesian machine used for the temporal localization method.

We used the BM-sliced machine with 3 columns per slice, as shown in figure 7.1. Each column is dedicated to a specific microphone m^1, m^2 or m^3 and is responsible for the temporal comparison of this microphone with the reference microphone m^0.

7.1.2 Simulations

To validate the algorithm and run the machine for different parameters, the machine has been implemented using our simulator presented in section 6.5.3.

Our sound simulator computes the sound mixture according to the location of the microphones and the source. It uses the sound propagation model described in [52] as:

$$m_t^i = \frac{1}{\sqrt{4\pi} \cdot d(c,i)} s_{(t-d(c,i)/v)} \tag{7.7}$$

Note that our sound simulator does not compute the reverberations. However, we added Gaussian noise to the signals to test the system robustness.

Localization map

In this section, a first example of a result provided by our localization method is presented. First, the different parameters of our system are explained.

The room size has been changed to $3.0m \times 3.0m$ in order to simulate the same room as the room used for the real world experiments.

Due to the attenuation model of our microphones which is close to the function described in equation 7.7, the energy in the signal significantly decreases as the source to microphone distance increases. We placed the microphones in the four corners of the room to maximize the performance of the system. Also, maximizing the inter-microphone distance increases the quality of the results. Therefore, in our standard configuration the microphones are placed in the corners of the room. Figure 7.2 illustrates the setup.

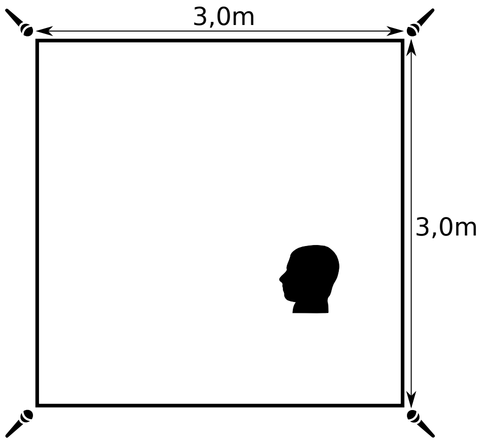


Figure 7.2: Setup of the room simulated to generate the sound for our experiments.

Moreover, the frame length which corresponds to the number of samples used for the localization is an important parameter for our system. It is also known as the recording time. Typically, in our tests, the size of the time interval used to localize the speaker is 30 samples.

In summary the parameters for our standard simulator setup are as follows:

- Room size: 3.0m x 3.0m
- Sound position: (2.0625,0.9375)
- Sound: F1 (sound file used given to the sound simulator)[1]
- Microphone positions: (0.1875,0.1875), (0.0,2.8125), (2.8125,0.0) and (2.8125,2.8125)
- Frame length: 30 samples

The parameters for the BM for the standard case are:

- Grid size: 8 x 8 tiles = 64 lines in BM
- Number of slices: 30 slices
- Columns per slices: 3 columns
- Counter max: 8 bits [2]
- Bits for probability representation: 8 bits
- Bits for offset: 8 bits
- Bits for attenuation fraction: 8 bits (4 bits before the comma and 4 bits after the comma)
- Shared random number generator: 1 LFSR per column

When running the Bayesian machine with the method presented in this chapter on our simulator, we obtain the localization map presented in figure 7.3. As one can see, the method works well as the maximum of the probability distribution is located in the cell where the sound source was positioned. Moreover, there are no cells close to the maximum which have a high probability value, hence the distribution is peaky.

[1]this sound file is recorded in an anechoic chamber without reverberations

[2]this represents the Resampling Threshold (RT) between each slice

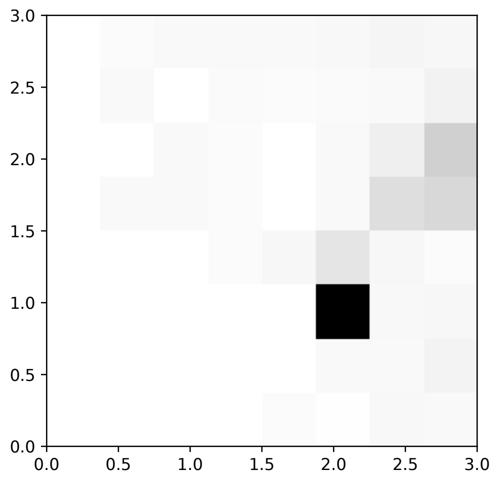


Figure 7.3: Localization map with for a frame of 30 samples.

7.1.3 VHDL implementation

The machine was implemented in the VHDL to simulate the circuit of the architecture and also run it on an FPGA. The architecture is similar to the one of the machine used for the SSL method with the Fourier transform presented in chapter 6. The core of the machine was reduced to 3 columns, as shown in figure 7.1. Thus, each slice has 3 columns where each of the 3 columns compares the value of the signal of it respective microphone m^1, m^2, m^3 with the reference microphone m^0.

However, the preprocessing module for the on-chip likelihood computation was modified and adapted to the new probabilistic model. The goal is to design a module which computes the likelihoods on chip according to the probabilistic model described in section 7.1. Remember, we want to compute the likelihoods as defined in equation 7.5:

$$P(m^i_{(t-\tau^i(c))}|\boldsymbol{m}^0 c) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(- \frac{(m^i_{(t-\tau^i(c))} - \frac{a(c,i)}{a(c,0)} \cdot m^0_t)^2}{2\sigma^2}\Big)$$

As in chapter 6, we tabulate the Gaussian distribution to avoid the on-chip computation of the exponential of the normal distribution. Nevertheless, we need to compute the following difference to address the right value of the ROM Gaussian:

$$m^i_{(t-\tau^i(c))} - \frac{a(c,i)}{a(c,0)} \cdot m^0_t \tag{7.8}$$

In the equation above, we can see that the $\tau^i(c)$ is applied to each microphone m^i to compute the right time index $t - \tau^i(c)$. If we do so, one might notice that we need to store a lot of microphone values since the interval for which we will need the samples is $T + 2\tau_{\max}$ with $\tau_{\max} = \max_{i\in I, c\in C} \tau^i(c)$ the maximal delay which can occur in this setup. However, if we apply the offset on the reference microphone 0, we will only have to store m^0 on the big interval. Hence equation (7.8) changes into:

$$m^i_t - \frac{a(c,i)}{a(c,0)} \cdot m^0_{(t-\tau^i(c))} \tag{7.9}$$

Note that the function $\tau^i(c)$ is previously computed and stored in a table as it depends on the microphone m^i and the position c. Moreover, the memory needs is decreased and can be illustrated as shown in figre 7.4.

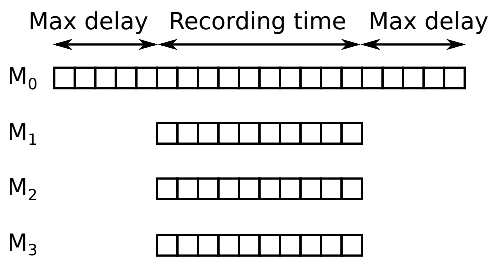


Figure 7.4: Different microphones used for the hardware implementation of the SSL method.

Thinking about the on-chip implementation of this computation, one can speed up the computation equation (7.9) by parallelizing the steps. Therefore, we rewrite the equation as:

$$\frac{a(c,0)}{a(c,i)} \cdot m^i_t - m^0_{t-\tau^i(c)} \tag{7.10}$$

Using this trick, the time index $t - \tau^i(c)$ for microphone m^0 and the multiplication $\frac{a(c,0)}{a(c,i)} \cdot m^i_t$ can be computed at the same time. Note that we tabulate the fraction $a(c,0)/a(c,i)$ to avoid to do this division on chip. This depends of the pair of microphones m^i and m^0 which is considered and also the position c. Therefore, as we have a total of 4 microphones including the reference microphone m^0, we have for each of the 3 pairs a table having the $\dim_x \times \dim_y$ lines in our memory block. Next, the global subtraction is calculated. The result of the subtraction is used to address the Rom

Gaussian to obtain the final likelihood value. The computation is performed in parallel for all the columns. Each column is running sequentially. The overall process is shown in figure 7.5.

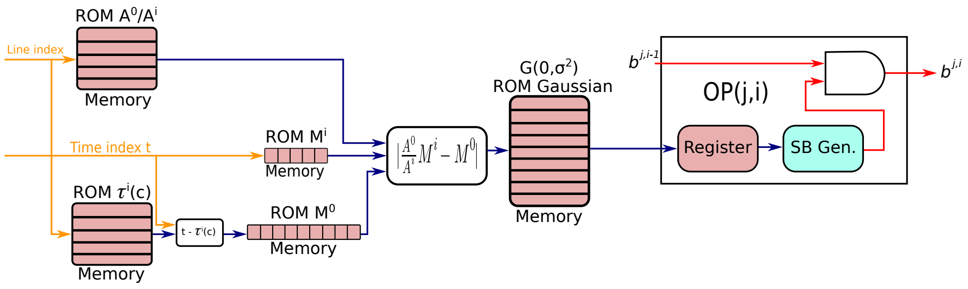


Figure 7.5: Preprocessing of the likelihood on-chip for the temporal localization method.

However, when doing so we change the variance σ^2 as we need to divide by the attenuation value $a(c, i)$ and hence this modifies the variance. Luckily, after studying the impact of this modification on the formula, we noticed that the result does not change as long as we are only interested by the maximum of the distribution. The distribution remains extremely peaky.

Due to this technique we are able to compute on chip the likelihoods. To reduce the circuit area and hence the power consumption, the computation is performed in fixed point. Both the offset and the attenuation fraction is represented on 8 bits. However, since the attenuation fraction is a comma value, it is represented with 4 bits before and 4 bits after the comma.

7.1.4 Experimentations

In this section, different experiments are presented to analyze the impact of the different parameters of the system on the overall localization performance. The first part of the presented results were obtained in simulations. The second part of the experiments are done using the VHDL implementation of the Bayesian machine.

Impact of the size of the frame

One very important parameter of our system is the frame size. In this section we present different results obtained while modifying the size of the frame, i.e. the recording time T. The number of samples T used by the systems directly impacts the performance of the system. When T increases, the system requires more time to process the data and hence more energy to perform that localization. However, it is critical to have a minimum amount of samples that are studied to localize the source as it significantly changes the quality of the result.

In figure 7.6 the probability distributions for different frame sizes are provided. We run the system with frames of size 1, 5, 10, 15, 20, 30, 50 and 100 samples. As the results show, a minimum amount of samples is necessary. When using 30 samples and more to localize the source, the quality is acceptable and the distribution becomes peaky. Therefore, our standard test case uses 30 sound samples.

However, as figure 7.6h shows, using 100 samples drastically increases the obtained localization quality. Nevertheless, on have to keep in mind that using 100 samples requires more than 3 times the time required by our standard case.

In conclusion, the most appropriate frame size has to be chosen according to the setup. It is a clear trade off between the computation time and the result quality.

Robustness to noise

As our homemade sound simulator does not compute the reverberations, we tested our system by adding noise to the signal. As we can consider reverberations as noise to our system.

We add independent Gaussian noise to each of the simulated signals recorded by the microphones. In this section we run the system for different values of sigma σ_{noise}.

Figure 7.7 provides localization maps for different values of σ_{noise} for $\sigma_{\text{noise}} = 1$, $\sigma_{\text{noise}} = 2$ and $\sigma_{\text{noise}} = 10$.

One can see that adding more noise to the signals decreases the quality of the result. Especially for $\sigma_{\text{noise}} = 10$ in figure 7.7c, the result has 2 peaks at 2 different distant places on the map with the first one at 255 and the second at 229.

In conclusion we can say that the system is robust to noise. However, too much noise increases the error rate of the method. Moreover, the added noise does not change the phase difference between the different microphone signals, which can be the case with real reverberations.

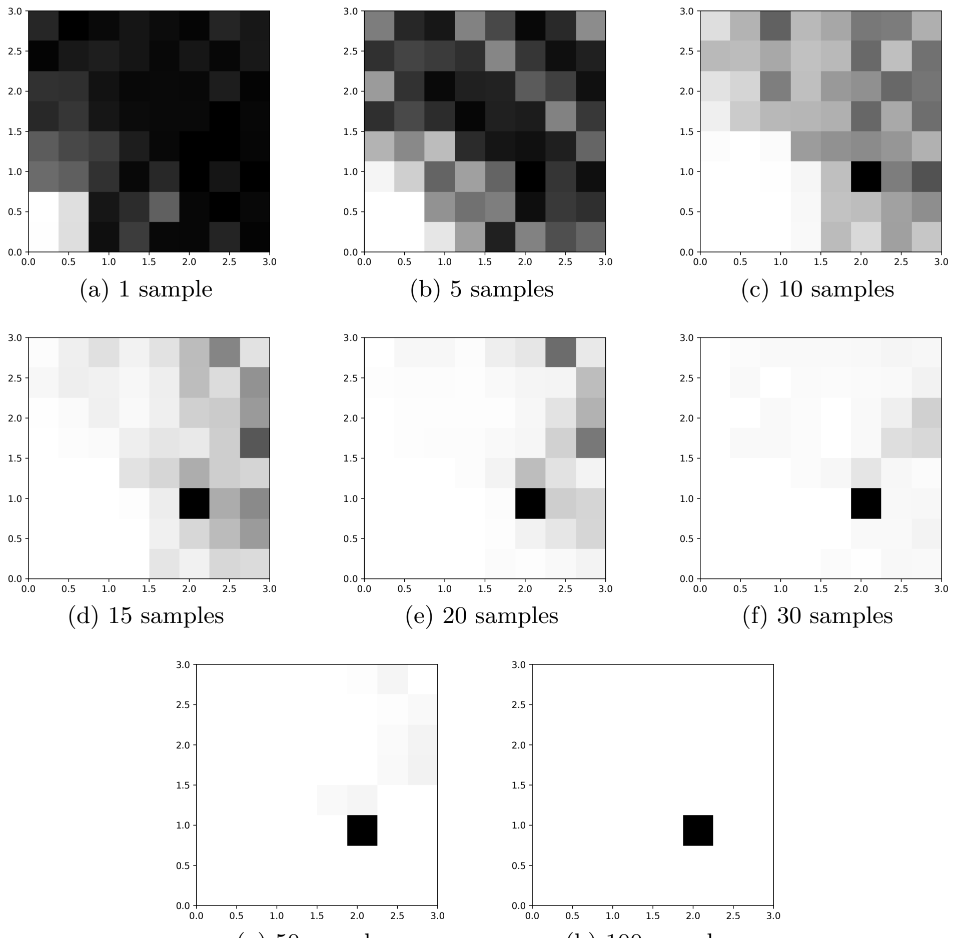

(a) 1 sample (b) 5 samples (c) 10 samples (d) 15 samples (e) 20 samples (f) 30 samples (g) 50 samples (h) 100 samples

Figure 7.6: Examples of localization when using different size of frames (recording time T).

Running in the real world

As in chapter 6, we confronted the system with the real world. Therefore, we modified the setup in our experimentation room. We recorded some signals with the setup as shown in figure 7.2.

In figure 7.8, a small portion of the signal is provided. The signals of the four microphones are each plotted in a separate color. This illustration shows how different the signals are although they recorded the same signal by the sound source. As the microphones are located in the corners of the

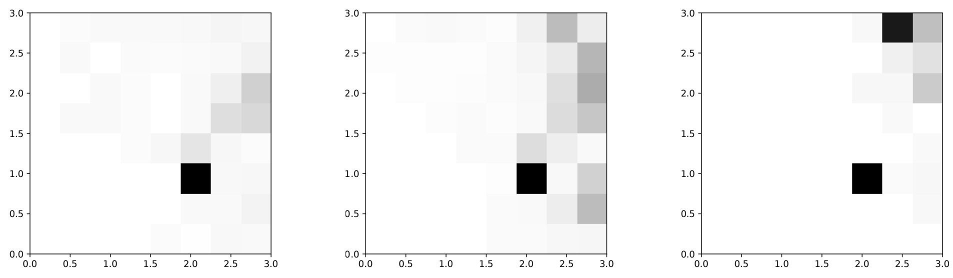

(a) Sigma = 1 (b) Sigma = 2 (c) Sigma = 10

Figure 7.7: Examples of localization when adding noise to the recorded signal.

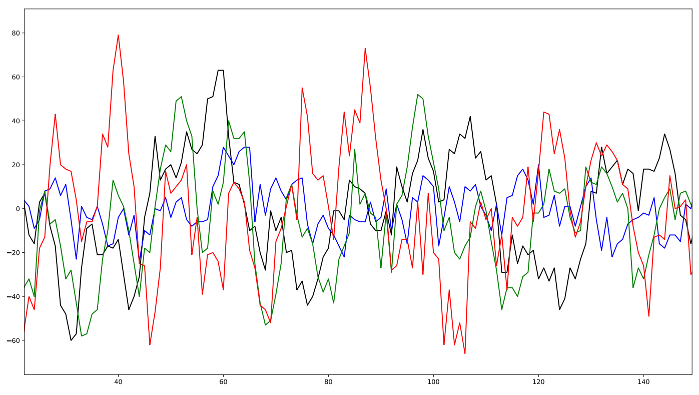


Figure 7.8: Microphone signals obtained in a real world recording.

room, it is obvious that the signals are shifted in time. However, our sound source localization method does not work in the real environment. This is due to the used attenuation, defined in equation (7.2). The attenuation model we encounter in the real world in our microphones is very different. As our method is extremely sensitive to the attenuation, it does not perform well. Especially because the gain on the microphones differ from each other and it is impossible to tune them precisely.

Therefore, it might be interesting to find a method which is not dependent of a precise attenuation model.

Circuit area

One of the main motivation for this localization method working without the Fourier transform is to reduce the circuit area and hence the power consumption. We studied the circuit area as we did for the SSL method presented in chapter 6.

Hierarchy	Resources FPGA (Comb ALUTs)	Memory RAM (bits)
top_SSL_FPGA	5219	44906
→ top_SSL	4342 (100%)	43008
→ top_preprocess	0 (0%)	0
→ top_BM1	4306 (99%)	43008
→ BM1_core	3383 (80%)	0
→ Control	92 (0.2%)	0
→ Memory_module	828 (19%)	43008

Table 7.1: Circuit area measurement of the system mapped on the FPGA for the SSL without Fourier transform.

In table 7.1 we provide the numbers about the circuit resources in terms of ALUTs and required memory. One can see that the preprocessing part (top_preprocess) is non-existing since we removed the Fourier transform by using this method. Moreover, numbers are much small than in the method of chapter 6. However, one has to consider that in this case the circuit has only three columns since each slice is dedicated to the temporal comparison on one specific time sample t. Therefore, the comparison is not very doable like that.

We modified the parameters of the circuit to generate the simulation for a circuit of 10 columns and 10 slices as the circuit of chapter 6 did have. Table 7.2 provides the measurements. In blue, the method of chapter 6, using the Fourier transform. In red, the method presented in this chapter which does not utilize the Fourier transform. The preprocessing part disappears in the new method since the Fourier transform is not used. Note that the on-chip likelihood computation is done in the memory module (memory_module). Both circuits implement a Bayesian machine with 64 lines, 10 columns and 10 slices. The total circuit area (top_SSL) has been reduced by approximately 30% and the memory needs were reduced by 55%. However, this came at a cost of the increasing memory module since the on-chip likelihood computation is more complex for this method since we compute a multiplication and several substractions. This made the memory module increase by approximately 60%.

Architecture	Comb ALUTs	RAM (bits)
top_FPGA	15039	381605
	11223	145257
→ top_SSL	13923	315432
	9996	143360
→ preprocess	4927	131112
	—	—
→ top_BM1	8931	184320
	9992	143360
→ BM1_core	7030	0
	7004	0
→ control	92	0
	100	0
→ memory_module	1803	184320
	2884	143360

Table 7.2: Circuit area measurement of the system mapped on the FPGA for the SSL without Fourier transform (in red) compared to the SSL with Fourier transform (in blue).

7.2 Probabilistic model without attenuation

In the previous sections, the probabilistic model using the attenuation was introduced and also analyzed. However, the testing in real environment showed us how sensitive the method is to the attenuation model since a small variance in the gain of the microphone changes a lot in the overall performance of the method. As it is very difficult to set the same gain for all microphones due to the analog part of the system, we studied another method which does not used the attenuation to localize the source.

This method does also entirely work in the temporal domain. It is based on the temporal comparison of two recorded signals. As the model with attenuation compares all microphones to one reference microphone, in this model the microphones signals are compared in pairs. The microphones of one pair are placed close to each other to compare their signals and extract an angular information.

We use 2 microphone pairs. More precisely, m^0 and m^1 form the first pair and m^2 and m^3 form the other pair.

In overall, given the signal recorded by the microphones m, the posterior probability distribution over $\boldsymbol{C}$ is computed as follows:

$$P(\boldsymbol{C}|\boldsymbol{m}) \propto \prod_{t=0}^{T} P(m^1_{(t-\tau^1_0(c))}|\boldsymbol{m}^0_t, \boldsymbol{C}) \cdot P(m^3_{(t-\tau^3_2(c))}|\boldsymbol{m}^2_t, \boldsymbol{C}) \tag{7.11}$$

The likelihoods $P(m^{i_1}_{(t-\tau^i(c))}|\boldsymbol{m}^0 c)$ and $P(m^{i_2}_{(t-\tau^{i_2}_{i_1}(c))}|\boldsymbol{m}^{i_1} c)$ are computed differently:

$$P(m^{i_2}_{(t-\tau^{i_2}_{i_1}(c))}|\boldsymbol{m}^{i_1} c) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{\left(m^{i_2}_{(t-\tau^{i_2}_{i_1}(c))} - m^{i_1}_t\right)^2}{2\sigma^2} \right) \tag{7.12}$$

To compute the distribution, equation (7.11) is evaluated for each point of the 64×64 source position grid and for a series of time samples t.

7.2.1 Modified setup

As mentioned above, the setup was changed to run this method as the microphones work in pairs and are closer located. Therefore, the setup is close to the one used in chapter 6. The microphones are located in the middle of the walls of the room with an inter-microphone distance equal to 20cm. Figure 7.9 illustrates the modified setup.

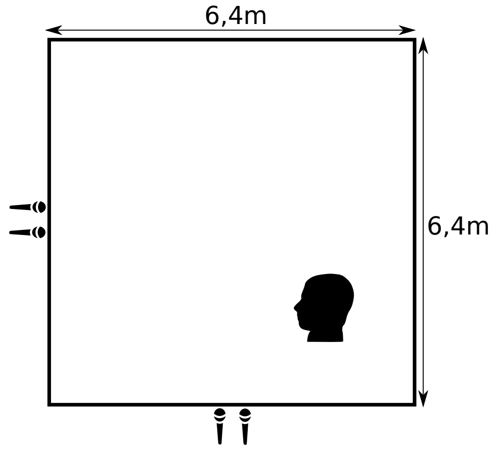
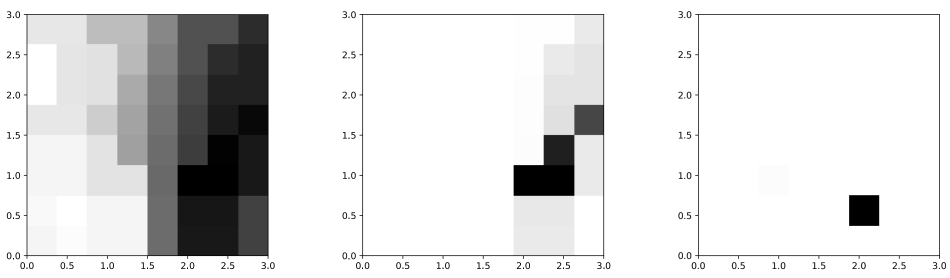


Figure 7.9: Setup of the room simulated to generate the sound for our experiments for the model without attenuation.

7.2.2 Localization map

In this section we show the result when running the Bayesian machine to perform the inference. Not that the following results are obtained in a setup in which we do not add any noise to the microphones.

Figure 7.10 shows the result for different recorded signals. In some portion of the signal, the localization method works a poor result quality, as shown in figure 7.10a. However, as for example in figure 7.10b, in other signal portion the system localizes the sound source much better and the obtained distribution gets peaky. Note that there are two cells in the grid with a maximum. This is due to parameters which have to be tuned to obtain a more peaky distribution. Moreover, as the figure 7.10c illustrates, some signal portions provide an extremely peaky distribution. Note that the max in this case is in the cell next to the real sound source which is located in the cell above ths black one.



(a) Start at 1100 (b) Start at 1800 (c) Start at 2500

Figure 7.10: Localization maps provided by method using temporal comparison without attenuation model.

7.3 Conclusion

In this chapter, two probabilistic methods to perform sound source localization in the temporal domain have been presented. The first one uses the attenuation of the signal as additional feature to localize the source. The probabilistic model of the first method has been introduced as well as the adjustments on the Bayesian machine to compute the probability distribution. Moreover, the VHDL implementation has been described, which also includes the on-chip likelihood computation method to be adapted. Experiments have been run to study the performance of this method. Also, the hardware implementation of the machine allowed to compare the circuit area to the method presented in chapter 6.

A second method has been introduced which does not use the attenuation of the signal in the model. Simulation results using this method have been presented. Although both presented methods provided promising results in simulation, their performance still needs to be shown in a real world environment.

Presenting these two methods showed that the Bayesian machine can be used for different applications and different models and easily be adapted to them. Moreover, the final implementation of the methods are very lightweight in terms of circuit area.

Chapter 8

Multi-Source localization in the temporal domain

Contrary to the methods presented in chapter 6 and chapter 7 were dealing with the mono-sound source localization task, this chapter presents a method for localizing multiple sound sources speaking at the same time. The original goal in our research project was to localize a sound source. However, while working on the sound source separation task, which will be presented in chapter 9, we noticed the need of having a sound source localization method able to deal with two sound sources. The method presented in this chapter is based on the mono-source localization method working in the temporal domain with attenuation presented in previous chapter, chapter 7. This concept was presented in an conference paper at the 2019 European Signal Processing Conference (EUSIPCO) in A Coruña, Spain [48].

This chapter is organized as follows. First, the localization method is presented. Second, the implementation in hardware is described. Third, simulations are done to show the performance of the system as well as it robustness to several setups.

8.1 Localization method

The method presented in this chapter is working in two major steps. In the first step, it performs sound source localization on a single (very) short time frame. This first step provides a probability distribution containing mainly one peak. It is run on several small time frames. In the second step, we fuse the localization information obtained by the first step for all the frames.

In the first step, we use the method with attenuation presented in chapter 7 and perform the localization as there would only be one sound source in the room. As the size of the time frames is very small (i.e. only a few samples), we assume that the signal contains the energy of one of both sources.

Let F be the total number of frames, T the total recording time for the frame expressed in number of samples, I the number of microphones used and J the number of sources placed in the room. The probabilistic variables are are the same as in the previous chapter with $\boldsymbol{M}$ the matrix of the stochastic variables associated to the sounds recorded by the microphones. Moreover, $\boldsymbol{C}^j = (C_x^j, C_y^j)$ is the probabilistic variables associated to coordinates of the sound source j with C_x^j.

The localization is made in small time intervals called frames. A frame is a set of T consecutive samples taken during this time window. It is assumed that during each frame the sound source does not move.

This section is split into the two main parts of the system. First, the probabilistic model used in each frame performing mono-source localization is described. Second, the fusion method is described.

8.1.1 Probabilistic model for single-frame single-source SSL

In this section the probabilistic model used in the first step of the localization system is described. The inference on this model is performed for each of the frames. The probabilistic model is based on the model presented in chapter 7. If you skipped chapter 7 while reading this document, please refer to chapter 7, more precisely to section 7.1 where the probabilistic model is described in a more detailed manner.

In short, in each frame f we compute the following distribution using the Bayesian machine:

$$P(\boldsymbol{C}|\boldsymbol{m}_f) \propto \prod_{t=0}^{T} \prod_{i=1}^{I} P(m^i_{f,(t-\tau_i(c))}|\boldsymbol{m}^0_f, \boldsymbol{C}) \tag{8.1}$$

where each conditional distribution $P(m^i_{f,(t-\tau_i(c))}|\boldsymbol{m}^0_f, \boldsymbol{C})$, also called likelihood, is given by:

$$P(m^i_{(t-\tau_i(c))}|c, \boldsymbol{m}^0) = \mathcal{N}(\frac{a(c,i)}{a(c,0)} \cdot m^0_t, \sigma^2) \tag{8.2}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes the Gaussian distribution with mean μ and variance σ^2. Source location c is estimated on a very short time-frame basis by finding the maximum value of the product in the right part of (8.1).

A keypoint of this method is that this approach to SSL works for a reasonable number of very short frames even if the microphone signals are composed of overlapping speech signals produced by two speakers speaking simultaneously from two different locations. Indeed, for a reasonable amount of such frames, the speech signal energy from one speaker has much larger energy than the signal from the other speaker, even if the speakers are speaking simultaneously. This is because a speech signal is a centered signal with fluctuating energy. For example, in a vowel, some successive samples have high energy, corresponding to a vocal fold pulse, and some successive samples have low energy, corresponding to the end of the vocal tract response to the pulse (before the next pulse reinjects energy). Therefore, at many occasions a few successive speech samples with high energy produced by one speaker correspond to a few successive samples with low energy produced by the other speaker, and vice versa. For very short time-frame containing such portions of speech signals, the localization method will work well. Moreover, due to the probabilistic approach and the facility of modelizing uncertainty using probabilistic models, one can say that our model does treat the second source present in the recorded signal as noise.

In practice, we have to find a trade-off between limiting the number of samples (for the above assumption to remain valid) and ensuring robust calculation of the posterior (8.1). Since the signals are recorded at 8-kHz sampling frequency and 8-bit quantization, our typical frame size of $T = 30$ samples is equivalent to 3.75 ms. This indeed corresponds to less than one period of voiced speech with fundamental frequency within 100-200 Hz. Note that this comes in contrast with the usual short-term frames used in the STFT-based SSL methods and in speech/audio analysis in general (typically within 20-30 ms).

Of course, frame-wise localization will not work well on frames where the signals from the two speakers are both of either same energy level. However, the fusion process described in the next section deals with this problem.

8.1.2 Fusion of frame-wise results for multiple-source SSL

The next step of the SSL method is the fusion of the information provided by the probabilistic model on very short time-frames, in order to provide multi-speaker localization.

First, as stated in the previous subsection, a frame-wise source location estimate $\hat{c}_f$ is computed for each frame f, as the candidate location which has the maximum posterior probability in that frame:

$$\hat{c}_f = \underset{c \in \boldsymbol{C}}{argmax} P(c|\boldsymbol{m}_f). \tag{8.3}$$

Then a global distribution map P_{glob} is created which counts for each candidate position $c \in C$, how many times it was selected as the most probable position $\hat{c}_f$ over all F frames. Typically, this counting process is done for a "reasonably large" number of frames, $F = 50$ in our simulations. Moreover, we spaced the frames with an interval of 70 samples (8.75 ms) in order to maximize the chance to capture different configurations of speech signal mixtures from the two speakers (see previous subsection). A bloc of microphone signal used to perform multi-source localization thus represents 0.625 s, which is quite reasonable for such a task.

As a result of choosing a sufficiently large number of very short time-frames, the obtained global distribution P_{glob} is in general very peaky and the locations of the two sources are clearly visible on the map (examples are provided in section 8.3.2). Final multi-source SSL then simply consists in selecting the two predominant peaks in the map. Note that, by doing so, we assume an *a priori* that there are two sources in the scene and we thus select two peaks. The alternative way is the unsupervised mode where a threshold is set for peak selection, see e.g. [76]. This latter approach has the advantage to automatically provide an estimate of the number of sources, which is generally unknown in practice, but it has the drawback of being sensitive to the threshold setting. However, as for our application, the two peaks corresponding to the two speakers are generally predominant in P_{glob}, the boundary between the supervised and unsupervised modes gets thin.

After each multi-source localization on a bloc of F frames, the map P_{glob} is reset to zero, and a new counting is processed on the next bloc.

8.2 Implementation on the Bayesian machine

In this section, the Bayesian machine used to perform the inference of the probabilistic model is presented. It is based on the architecture presented chapter 6. Since the architecture uses stochastic computing, the likelihoods which are multiplied in the inference equation (8.1) are represented as stochastic bit streams and multiplied using AND-gates.

For this aim, the BM is structured as a $w \times (T \times (I - 1))$ matrix, where $w = card(C_x) \times card(C_y)$ is the number of source location candidates on the grid. Figure 8.1 shows a schematic representation of the BM architecture for 3 lines and 3 columns (for all 3 microphones i, $i + 1$ and $i + 2$ at a given

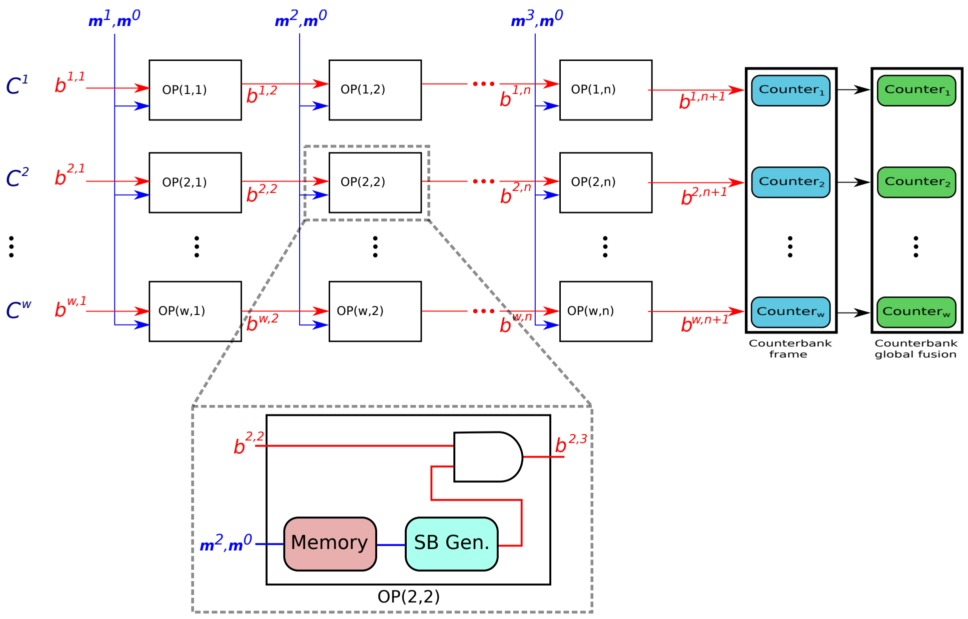


Figure 8.1: Simplified representation of the Bayesian machine architecture used to compute the inference equation for SSL and the fusion.

time sample t). Each line of the machine computes (8.1) for a specific value of source location c. In each column one microphone $i \in [2, I]$ at a given time $t \in [1, T]$ is compared to microphone 1 which leads to the evaluation of $P(m^i_{(t-\tau_i(c))}|c\boldsymbol{m}^0)$. An AND-gate is present in each OP-block, which represents the multiplication in stochastic computing. In our simulations we used $I = 4$ microphones. The machine had thus $N \times (I-1) = 30 \times (4-1) = 90$ columns and $1,024$ lines considering a grid of 32×32 possible source locations. At the end of each line, counters (in blue) count the number of "1"s in the output stochastic bit stream.

However, the fusion method is new in the architecture and needs to be added in our circuit design. As the fusion method was developed with the final architecture in mind, it was kept as simple as possible. Therefore, an additional counterbank (in green in the figure) was added to the existing one (the blue counters). At the end of each frame, the line with the maximum counting activates the corresponding counter in the further global fusion counterbank (in green), which implements the fusion process explained in section 8.1.2, and frame-wise (blue) counters are reset to zero.

8.3 Simulations

In this section, we present the simulations conducted to evaluate the SSL method.

8.3.1 Setup

Simulations have been processed with a 6.4 m $\times$ 6.4 m room discretized in 20 cm $\times$ 20 cm tiles, leading to a grid of $32 \times 32 = 1{,}024$ candidate 2D source positions (x, y) for c.

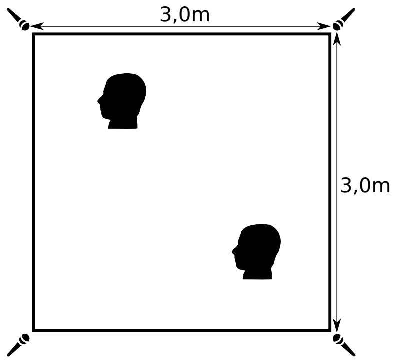


Figure 8.2: Simulated room setup for our multi-source localization method.

As shown in figure 8.2, $I = 4$ microphones have been placed in the room. Each pair was located in the middle of two adjacent walls. The inter-microphone distance was 20 cm. For our first simulations, the sources have been placed in the middle of the grid cells indexed by $(8, 24)$ and $(24, 8)$, as illustrated in figure 8.2.

Utterances from the TIMIT database [53], resampled at 8 kHz, have been used as speech material. To generate the multichannel mixture signal recorded at the microphone array, a simple spatial sound simulator has been used which implements the spherical wave model as defined in equation (1) in [52] (see equation 7.7).

Let us remind that in our simulations, we performed the "individual" frame-wise localization on $F = 50$ frames of $T = 30$ time samples each (i.e. 3.75 ms) and the multi-source localization with frame-wise results fusion is performed on successive blocs of 0.625 s.

As previously mentioned, the design of our localization method was driven by the low-power stochastic architecture that computes the inference of our probabilistic model.

8.3.2 Localization performance

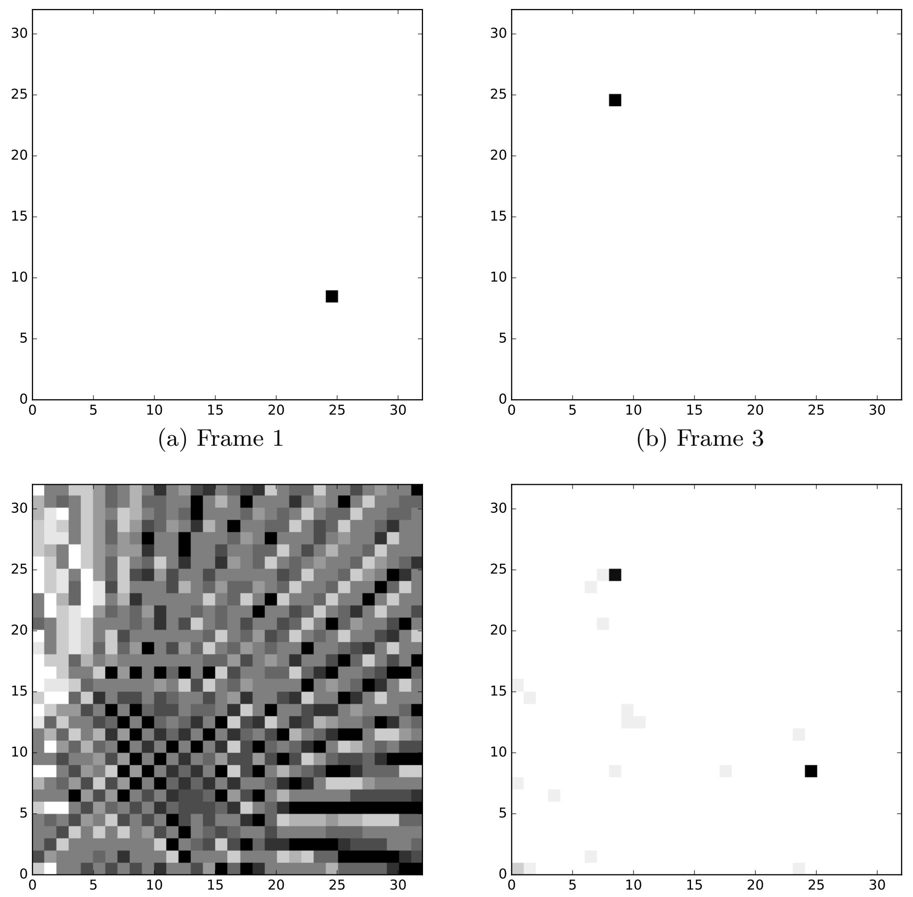
Let us first present the localization results obtained on single very short time-frames, before showing the final distribution obtained by the fusion process explained in section 8.1.2.

Figure 8.3 shows the posterior distribution $P(\boldsymbol{C}|\boldsymbol{m})$ obtained for 3 different frames, as a function of candidate 2D source location on the 32×32 room grid. The goal is to detect the two sources, which in the grid cells are indexed by $(8, 24)$ and $(24, 8)$. The darker a cell is, the higher the probability for this cell (in other words, 1 is black and 0 is white). For the purpose of well illustrating the behavior of the mehod, we selected 4 frames with quite different, though representative, type of results. Indeed, as one can see, frame 1 and frame 3 each locate perfectly one of the two sources. However, in some frames, the localization does not provide a clear maximum position, as shown in frame 29 and frame 46. However, most frames provide results similar to frame 1 and frame 3.

When fusing the information from 50 frames following the strategy explained in section 8.1.2, the resulting map shows two clear maxima located at the actual source positions, as seen in figure 8.3(d). The map has two black dots which correspond to the two source locations. Some cells are in light grey (representing a low probability) which shows that these cells obtained the maximum of the posterior distribution $P(\boldsymbol{C}|\boldsymbol{m})$ for some frame(s). The localization method is robust in the sense that the contrast between the detected/actual sources locations (black dots) and the other cells with non-zero probability (light grey) is strong enough. Quantitatively, for this example, the source located at $(24, 8)$ is detected by the frame-wise posterior maximum in 16 frames, and the source at $(24, 8)$ is detected in 15 frames. All other cells (in light grey) are counted at most 3 times as frame-wise maximum. This means that the remaining randomly-positioned 19 frame-wise maxima do not impact on the final result. This shows the importance to find a good setting for the frame size T and for the number of frames F.

In summary, due to the fusion process over the F frames, a robust blind source localization method is obtained.

As briefly mentioned in section 8.1.2, because of this high contrast between the two main peaks and the other non-zero values in the global localization map, the system could automatically determine the number of sources by counting the number of peaks over a certain threshold. However, this was not systematically tested in the present simulations.



(a) Frame 1 (b) Frame 3

(c) Frame 29 (d) Final distribution map after fusion

Figure 8.3: (a) to (c): Posterior distribution map obtained for 3 very short time-frames of a given 50-frame bloc. (d) Final distribution map after fusion over 50 frames. The two black squares correspond to the actual positions of the two sources.

8.3.3 Robustness to various source locations

To analyze the performance of the localization method more quantitatively, simulations with 2 sources were conducted for various source locations. In total, 12 different setups have been evaluated with different randomly chosen source positions. The true source position has been compared to the estimated sources positions.

Results show that out of the 12 setups, in 4 cases, both sources were located at the right position in the grid. Moreover, in 6 out of the 12 cases, one source was estimated at the true position and the other one was estimated in an adjacent cell in the grid. Finally, in 2 cases, both sources were positioned in a neighbouring cell next to their respective true position. In practice, considering that one cell is 20 cm wide, the maximum error done by the system is about 28 cm (in case of a diagonal neighbor cell). Analyzing the error distance over the 24 estimated positions (2×12), an overall average error distance of 0.12 m between the estimated position and the true position is obtained.

8.4 Conclusion

In this chapter, a multiple sound source localization method has been presented. The method works in two main steps. First, we use the method described in the previous chapter (chapter 7) to localize one sound source in very short time frames (only a few signal samples are required 3.75 ms). Second, the results of the different frames are fused to obtain the global probability distribution, which includes the different sources. The method has been implemented on the Bayesian machine. The fusion process has been added to the existing architecture. Note that the fusion process is very simple to implement in hardware and thus not requiring a lot of resources. Simulations have been presented to show the performance of the system and the robustness in various setups, which is based on the peakiness of the global localization map.

Chapter 9

Source separation

The idea of this thesis is to work on two different applications in the signal processing area, namely the sound source localization and the sound source separation. Both applications require different kinds of algorithms, which results in another type of machine for us.

Opposed to the previous chapters dealing with the sound source localization application, i.e. chapter 6, chapter 7 and chapter 8, this chapter presents a way to perform audio source separation. The chosen method described in this chapter is based on a MCMC sampling method: the Gibbs algorithm.

The presented method is working in two main parts. First, the sources need to be localized. Second, the separation is computed knowing the source positions. A solution for the first step has already been described in chapter 8. This method delivers the source positions to the separation method, which is the second part of the system.

This chapter is organized as follows. First, the general model is presented before describing how we apply the Gibbs sampler on it. Then, the Bayesian sampling machine for this specific application is introduced. Moreover, high level simulations are shown. Finally, simulations of the machine dedicated to this application are described.

9.1 General model

In this section we introduce the probabilistic model used for the source localization and separation method. Therefore, we first need to define the probabilistic variables:

- $\boldsymbol{M}$: the matrix of the stochastic variables associated to the sounds recorded by the microphones with M_t^i the stochastic variable associated to microphone i at time t and m_t^i its value.

- $\boldsymbol{S}$: the matrix of stochastic variables associated to all the sounds emitted by the sources with S_t^j is the stochastic variable associated to the sound emitted by source j at time t and s_t^j its value. Each value of S is discretized on $g = \dim_s$ values.
- $\boldsymbol{C}^j = (C_x^j, C_y^j)$: the probabilistic variables associated to coordinates of the sound sources. C^j denotes the position of source j with $C_x^j \in [0, \dim_y]$ and $C_y^j \in [0, \dim_y]$ with $\dim_x$ and $\dim_y$ the dimensions of the room in the x and y axis.

Let T be the total recording time on which we are working expressed in number of samples, I the number of microphones used and J the number of sources placed in the room. The values of the stochastic variables associated to the sounds recorded by the microphones are discretized between $-m_{\max}$ and $m_{\max}$ and so, the values of the m_t^i are taken in this discrete set : $m_t^i \in \{-m_{\max}, \ldots - 1, 0, 1, \ldots, m_{\max}\}$. The same applies to the stochastic variables associated to the sound emitted by the sources: $s_t^i \in \{-s_{\max}, \ldots - 1, 0, 1, \ldots, s_{\max}\}$.

Given a source location c^j, and the position of microphone i, the time difference expressed in number of samples:

$$\delta_i^j = d(j,i) \cdot \frac{F_s}{v} \tag{9.1}$$

where $d(j,i)$ is the distance between source j and microphone i, v is the sound celerity, and F_s is the sampling frequency.

We can compute the maximal time difference $\delta_{\max}$, which is the maximal time difference occurring between any microphone and any source in the chosen setup. In practice, $-\delta_{\max}$ denotes the time before which no signal will ever reach any microphone at time 0:

$$\delta_{\max} = \max_{i,j \in I \times J} \delta_i^j \tag{9.2}$$

Moreover, as we recorded the microphone signals on the time interval $\{0, ..., T\}$, we have to take into consideration the source signals on the time interval $\{-\delta_{\max}, ..., 0, ..., T\}$ since at time 0 we will receive the source signals emitted at time $-\delta_{\max}$. In other words, when using the free field model, only the values of $s^j_{t-\delta_i^j}$ should be considered when computing M_t^j. This means that if $t - \delta_i^j < -\delta_{\max}$, the source j is not taken into account as the sound can physically not be received by microphone i.

The joint distribution of our model $P(\boldsymbol{S},\boldsymbol{C},\boldsymbol{M})$ can be written as:

$$P(\boldsymbol{S},\boldsymbol{C},\boldsymbol{M}) = P(\boldsymbol{S})P(\boldsymbol{C})P(\boldsymbol{M}|\boldsymbol{S},\boldsymbol{C}) \tag{9.3}$$

When discretizing the variable M, we can rewrite the decomposition in the equation 9.3 as:

$$\begin{aligned} P(\boldsymbol{S},\boldsymbol{C},\boldsymbol{M}) &= P(\boldsymbol{S})P(\boldsymbol{C}|\boldsymbol{S})\prod_{i=1}^{I}\prod_{t=1}^{T}P(M_t^i|\boldsymbol{S},\boldsymbol{C}) && (9.4)\\ &= P(\boldsymbol{S})P(\boldsymbol{C})\prod_{i=1}^{I}\prod_{t=1}^{T}P(M_t^i|\boldsymbol{S},\boldsymbol{C}) && (9.5)\end{aligned}$$

To compute the equation 9.4, we assume that we have no knowledge about the sources S_t^j, which leads to a uniform distribution on $\boldsymbol{S}$. Considering $P(\boldsymbol{C})$ we also define it as a uniform distribution. Since the microphone value is the mixture of the different signals emitted by the different sources and we assume it to follow a normal distribution, we define $P(M_t^i|\boldsymbol{c},\boldsymbol{s})$ as follows:

$$\begin{aligned} P(M_t^i|\boldsymbol{c},\boldsymbol{s}) &= P(M_t^i|\boldsymbol{c}, s^0_{t-\delta_i^j}, ..., s^j_{t-\delta_i^j}, ..., s^{J-1}_{t-\delta_i^j}) && (9.6)\\ &= \mathcal{N}(m_t^i; \sum_{j=0}^{J} a(c^j,i)s^j_{t-\delta_i^j}, \sigma) && (9.7)\end{aligned}$$

with $a(c^j,i)$ the attenuation factor between a source j and a microphone i. This function is defined according to equation (1) in [52] (see equation 7.7). The signal on microphone i is the weighted sum of all the emitted signals with a given noise σ.

The boundary condition is as follow:

$$s^j_{t-\delta_i^j} = 0 \quad \text{if} \;\; t-\delta_i^j < -\delta_{\max} \tag{9.8}$$

9.2 Sampling using Gibbs

The goal in this application is to separate the sources which are recorded as a mixture by the microphones. Therefore, we need to sample the distribution $P(\boldsymbol{C},\boldsymbol{S}|\boldsymbol{m})$ to obtain samples for the signals and for the positions of the sources using the model previously described. Using the Gibbs algorithm, one could sample the distribution $P(\boldsymbol{C},\boldsymbol{S}|\boldsymbol{m})$, which samples the positions and the signals of the sources at the same time. However, for several reasons, we decided to focus on a sub-problem which is $P(\boldsymbol{S}|\boldsymbol{c}^\star,\boldsymbol{m})$ where

$\boldsymbol{c}^\bigstar$ are the positions of the sources estimated by our multi-source localization method presented in chapter 8. Some other problems such as sampling from $P(\boldsymbol{C}, \boldsymbol{S}|\boldsymbol{m})$ directly or from $P(\boldsymbol{C}|\boldsymbol{s}^\bigstar, \boldsymbol{m})$ were not considered in this work. The different reasons will be discussed at the end of this chapter.

Using the Gibbs algorithm to sample the distribution $P(\boldsymbol{S}|\boldsymbol{c}^\bigstar, \boldsymbol{m})$ does require to draw a sample from

$$P(S_t^j|\boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\bigstar, \boldsymbol{m}) \tag{9.9}$$

for each $j \in \{0, ..., J\}$ and for each $t \in \{0, ..., T\}$ where $\boldsymbol{s}_{\neq t}^{\neq j}$ is the set of variables including all variables $\boldsymbol{s}$ except the value $\boldsymbol{s}_t^j$. Using our probabilistic model, more precisely the joint distribution defined in equation 9.3:

$$\begin{aligned} P(S_t^j|\boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\bigstar, \boldsymbol{m}) &= \frac{1}{z} P(S_t^j, \boldsymbol{s}_{\neq t}^{\neq j}) P(\boldsymbol{c}^\bigstar) P(\boldsymbol{m}|S_t^j, \boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\bigstar) \\ &= \frac{1}{z'} \prod_{i=1}^{i=I} \prod_{l=1}^{l=T} P(m_l^i|S_t^j, \boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\bigstar) \end{aligned} \tag{9.10}$$

since $P(S_t^j, \boldsymbol{s}_{\neq t}^{\neq j})$ is uniform as we do not have any further information on this distribution. However, $P(\boldsymbol{c}^\bigstar)$ is a peaky distribution considered to be a dirac in the locations of the sources such as $P(\boldsymbol{C}) = \delta_{c=\boldsymbol{c}^\bigstar})$.

Since most of the terms of the product found in equation 9.10 are constant for all the values taken by S_t^j, this allows to reduce equation 9.10 to:

$$P(S_t^j|\boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\bigstar, \boldsymbol{m}) = \frac{1}{z'} \prod_{i=1}^{i=I} \mathcal{N}(m_{t+\delta_i^j}^i; a(\boldsymbol{c}^{\bigstar j}, i) \cdot s_t^j + \textstyle\sum_{l \neq j} a(\boldsymbol{c}^{\bigstar l}, i) \cdot s_{t+\delta_i^j - \delta_i^l}^l, \sigma^2) \tag{9.11}$$

Equation 9.11 can be interpreted as follows: to get the information on S_t^i we use the signal of each microphone i at time $t + \delta_i^j$. Also, we have to consider the other sources l at time $t + \delta_i^j - \delta_i^l$ and sum all of them to obtain the likelihood on $M_{t+\delta_i^j}^i$

Moreover, looking at the boundary conditions, equation 9.11 is not valid when:

- $t + \delta_i^j < 0$: because the sound of source j will reach microphone i before the recording starts
- $t + \delta_i^j > T$: because the recording ends before the sound emitted by the source j could reach it

As mentioned in section 9.1, it is crucial to ensure:

$$t + \delta_i^j - \delta_i^l < -\delta_{\max} \qquad \forall j, l \in \{1 \dots J\} \tag{9.12}$$

The separation method using Gibbs computes equation 9.11, for all values of S (for all $j \in \{0, ..., J\}$ and all $t \in \{0, ..., T\}$). At each Gibbs iteration, it takes into account the current values for all elements of S and computes the equation 9.11. Each entire iteration over all the variables of S is called a *sweep*.

9.3 Bayesian Sampling Machine adapted to perform the source separation

In this section, we describe how the Bayesian sampling machine (BSM) has been adapted to perform the sound source separation. The machine we use for this task is different from the architecture we used for the localization problem in the previous chapters. It was introduced in section 4.3.

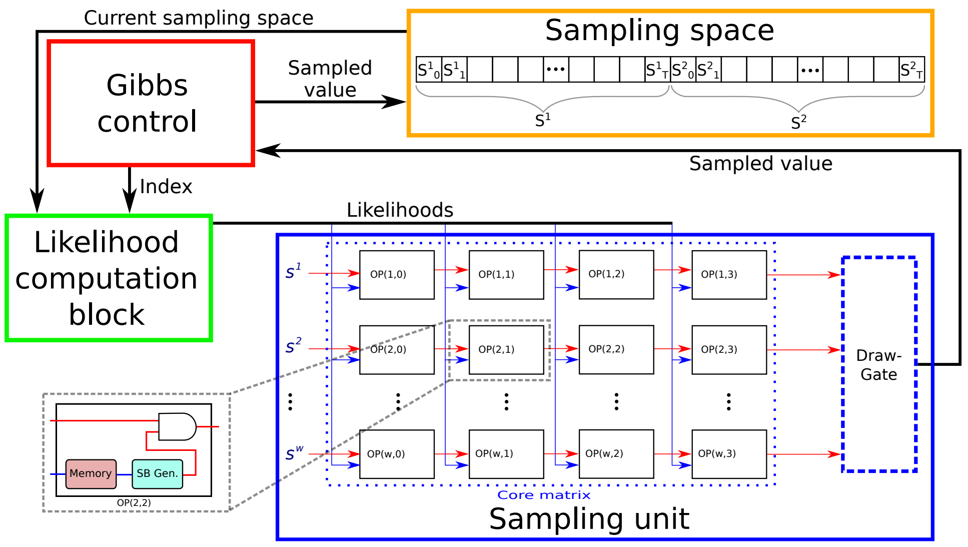


Figure 9.1: Architecture of the Bayesian sampling machine (BSM) adapted to the source separation problem.

Figure 9.1 shows how the architecture of the BSM was adapted to the source separation problem. In orange one can see the module, which is mainly a memory block storing the current values in the sampling space.

This memory is initialized with random numbers. Once one value has been sampled from the distribution 9.11, the value is stored in the memory in orange.

The sampling unit, in blue, is the heart of the machine, which is responsible to compute equation 9.11. The core matrix is close to the Bayesian machine we used for the sound source localization problem. However, in this case the unit does not compute the overall probability distribution. It draws a value of the distribution. This is the reason why there are no counters at the end of the lines. They are replaced by the draw-gate, explained in section 4.3.1. Once the value has been sampled by the sampling unit, the value is given to the Gibbs control unit (in red), which stores it in the sampling space memory (in orange). The Gibbs control block sets which S_t^j we want to sample and at each step the likelihoods of equation 9.11 are computed by the likelihood computation unit (in green) which calculates the likelihoods depending of the current values in the sampling space before giving them to the sampling unit (in blue). The likelihoods are stored in the memory in the right OP blocks in the sampling unit. Once all the likelihoods are computed, the sampling unit starts its computation.

This iteration is done for every variable S_t^j. Once all the variables in the sampling space have been sampled, one sweep of the Gibbs algorithm is completed. Typically, the Gibbs algorithm performs several sweeps before converging.

9.4 Simulations

We have run simulations of our system to evaluate its performance. The simulations were done in two steps. First, we implemented the algorithm in C++/Python to test the system with different sets of parameters. In a second step, we have run the Bayesian Sampling Machine adapted as explained in section 9.3 to perform the source separation.

9.4.1 Simplified sound simulator

To generate the sound mixture of different microphones, we used our sound simulator, which computes the sound signals according to the location of the microphones and the sources. It uses the sound propagation model described in Equation (1) in [52] (see equation 7.7). The signals of the different sources are added to generate the final signal recorded by the microphones. Note that our simulator does not compute reverberations. However, in some of our simulations, we added Gaussian noise to the signals to test the system robustness.

9.4.2 Standard experimentation setup

To test our source separation system, we use a standard set of parameters defined below.

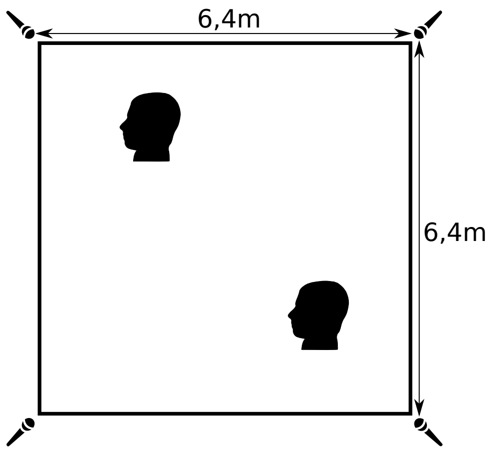


Figure 9.2: Position of the speakers and the microphones to simulate the sound for our experiments.

For the sound generation and the audio pre-processing, our standard case contains the following parameters:

- Room size: 6.4m x 6.4m
- Number of sources: 2
- Source positions: (1.2 , 4.4) and (4.4 , 1.2)
- Sound for source 1: F1 (utterances from the TIMIT database [53])
- Sound for source 2: F2 (utterances from the TIMIT database [53])
- Microphone positions: (0.0 , 0.0), (0.0 , 6.4), (6.4 , 0.0) and (6.4 , 6.4)
- Sampling rate: 8 kHz
- Recording time: 80,000 samples (10 seconds)
- Signal discretization: $g = \dim_S$ ={-128,...,0,...,+128}

Figure 9.2 shows the setup use for our standard case. Note that we use the same setup as for the multi-source localization method (presented in chapter 8) since both systems will interact and run in the same setup. First we localize the sources before starting the separation.

9.4.3 Sampling space

In this section, we want to illustrate the search space in which the Bayesian sampling machine is sampling to find the signals.

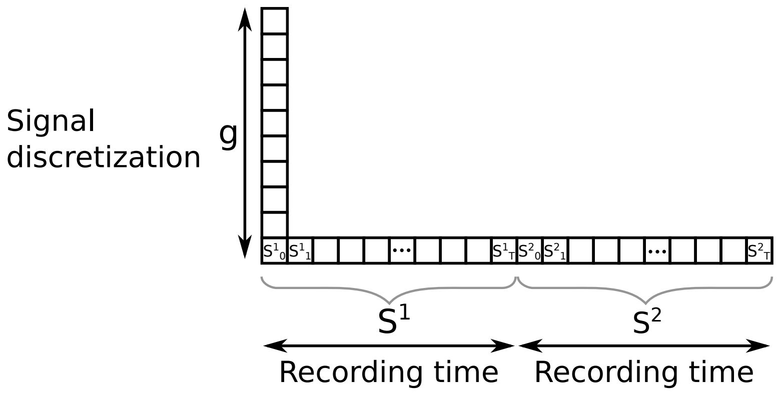


Figure 9.3: Visualization of the sampling space.

As stated in the previous section, section 9.4.2, in our standard experiment setup, we separate 10 seconds of signal. Since our sample rate is 8 kHz, we obtain 8000 samples per second. This means we have 80000 samples of signal to compute for each source. We have two sources in our setup. Therefore, we have 160000 samples to compute for both sources.

The signals are discretized on g values between {-128,...,0,...,+128}. In total our search space contains $160,000 \times 256 = 40,960,000$ possible values. Figure 9.3 illustrates the different dimensions of the sampling space.

Initially, the sampling rate was 16 kHz. As the sampling rate directly defines the size of the sampling space, we reduced the sampling rate to 8 kHz, which is commonly used for telephones as it is still suitable for human voice. Moreover, we reduced the signal discretization to 8-bit signed as this also reduces the dimension of the search space.

9.4.4 Separation result

In this section, we present a first result of a source separation. The setup for this experiment is as described in section 9.4.2. We did not add any noise to the recorded signal as we want to study the separation performance of the system. The performance when adding noise will be studied in an upcoming section.

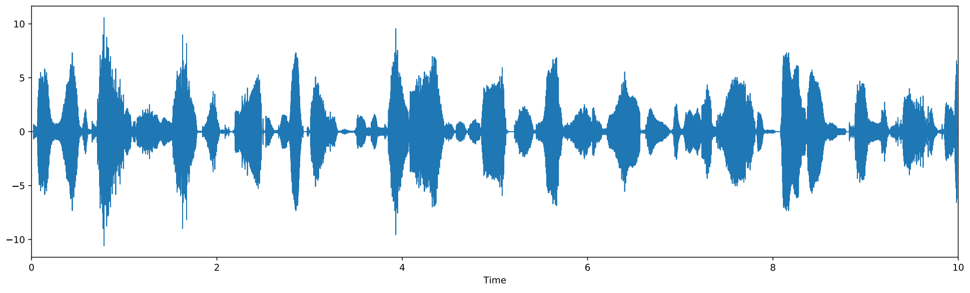


Figure 9.4: Signal recorded by microphone 1.

Figure 9.4 shows the 10 seconds of signal recorded by microphone 1. In this signal, both sources speak simultaneously. One can see the that there is no noise added to the signal recorded by the microphone since at some points the signals is equal to 0 as both sources do not speak (i.e. signal at t=3 seconds).

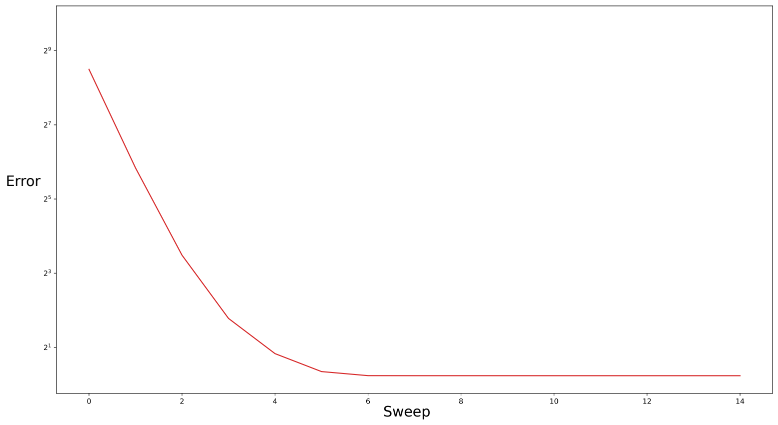


Figure 9.5: Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps.

In total, we ask the system to compute 15 sweeps. In other words, the Gibbs algorithm iterates 15 times over the entire set of searched variables. As the number of iterations increases, the quality of the results becomes better. Figure 9.5 illustrates the error depending on the number of iterations. The plot shows the mean squared distance between the original source signal and the sampled signal. One can see that the error significantly decreases during the first iterations and starts to be constant after sweep 6. The improvements in the signal due to the sweeps 4,5 and 6 are very limited compared to the first 4 sweeps.

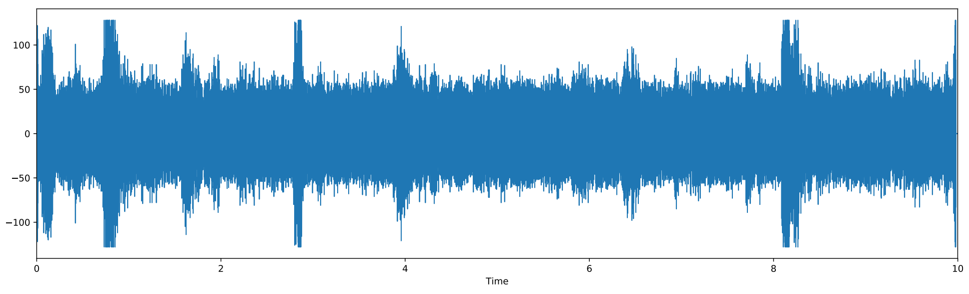


Figure 9.6: Reconstructed signal of source 1 after sweep 0.

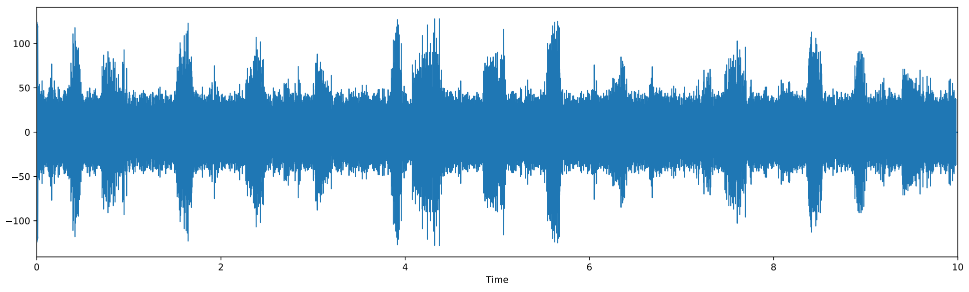


Figure 9.7: Reconstructed signal of source 1 after sweep 1.

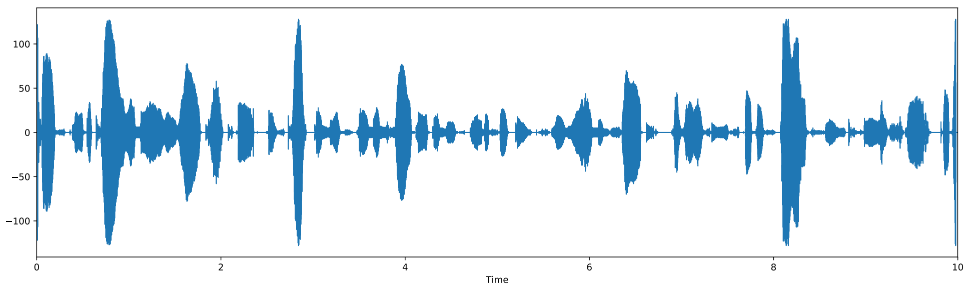


Figure 9.8: Reconstructed signal of source 1 after sweep 7.

Figure 9.6 shows the signal of source 1 estimated by the separation system after one sweep by the Gibbs algorithm. One can see a lot of noise in the signal. After the second sweep, the noise is reduced, as shown in figure 9.7.

Finally, after only 7 sweeps, the noise is completely removed from the signal of source 1, as illustrated by figure 9.8.

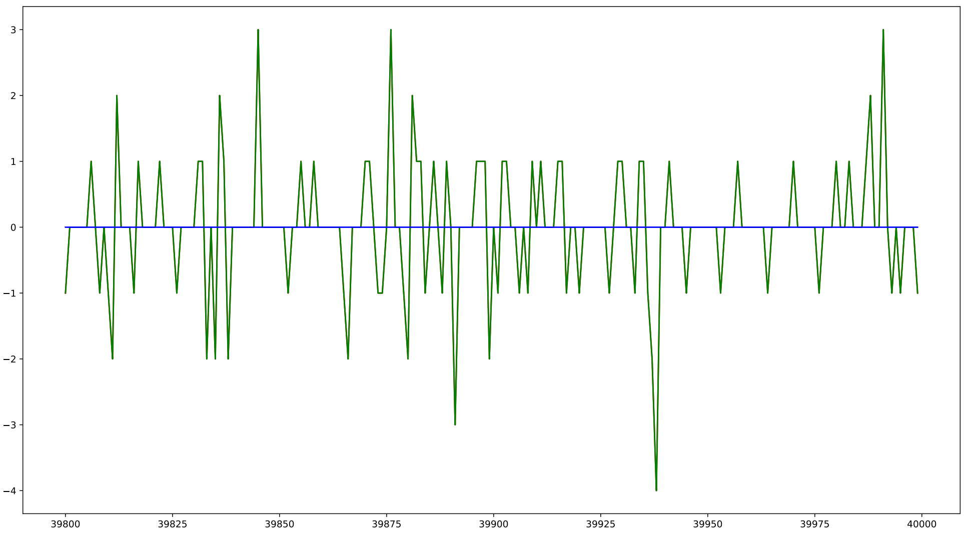


Figure 9.9: Quality of the estimated result: original signal in red, estimated signal in green and blue the difference between the original and the estimated signal.

To illustrate the quality of the final result, figure 9.9 plots 200 samples of the original signal and the estimated signal. In red the original signal, in green the estimated signal and in blue the difference of them. As both the original and the estimated signal totally overlap, one can not see the original signal (in red).

In total, the computation time required to separate the 2 sources on 10 seconds of signal is 6.80 seconds. This is the time needed to perform the 15 iterations of Gibbs. However, the signals were already perfectly separated after 7 sweeps. In practice, one could stop the computation after the first 7 sweeps. Moreover, when considering that the system only needs 7 sweeps, the total computation time would be 2.67 seconds. Note that this is much faster than real time since we are estimating 10 seconds of signal.

9.4.5 Separation result with noise

As our sound simulator does not add any reverberation in the computed signal, we tested the system by adding Gaussian noise to the signal of the microphones. This also shows the robustness of the system.

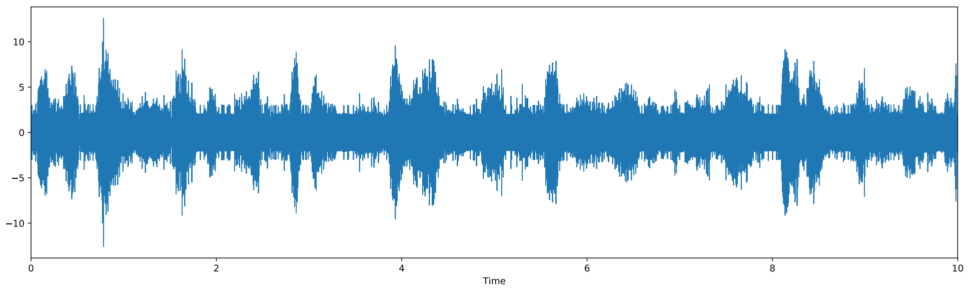


Figure 9.10: Signal recorded by microphone 1 with added Gaussian noise.

Figure 9.10 shows the 10 seconds of the signal recorded by microphone 1. One can see that the signal does never get completely to 0 (silence) as we have added some Gaussian noise to the signal.

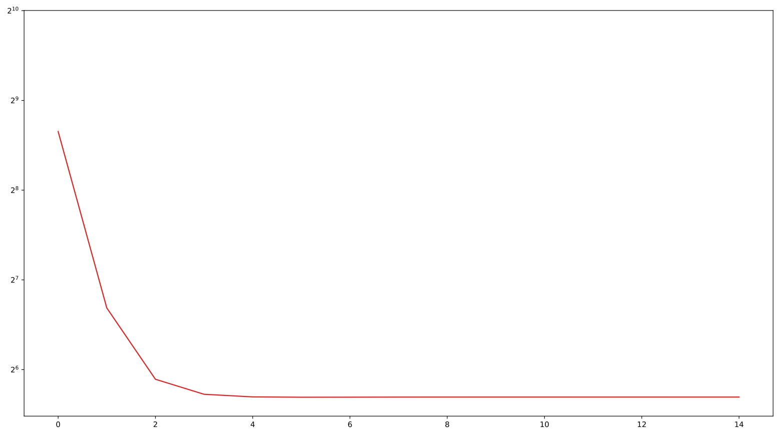


Figure 9.11: Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps when adding Gaussian noise to the recorded signal.

We run the Gibbs algorithm for 15 sweeps. Figure 9.11 shows the evolution of the mean squared distance between the original source signal and the estimated signal as a function of the number of sweeps. The error decreases very quickly and already after 5 sweeps the final result is obtained.

Figure 9.12 shows the signal of source 1 estimated by the separation system after 15 sweeps (same signals as after 5 sweeps as the algorithm already converged). One can see that the signal still has a lot of noise. This obtained noise is due to the Gaussian noise added to the system when simulating the recorded signals.

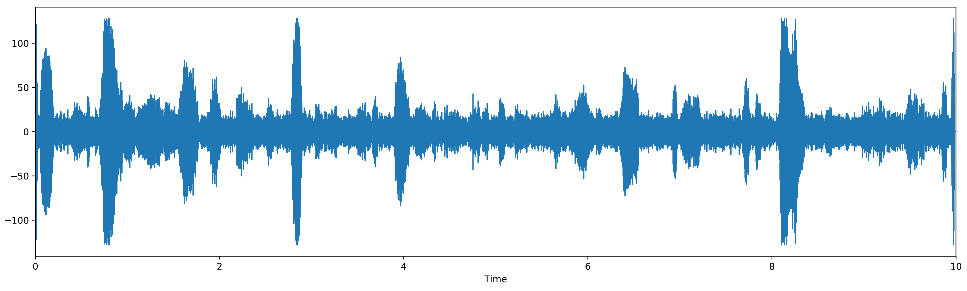


Figure 9.12: Reconstructed signal of source 1 after sweep 15.

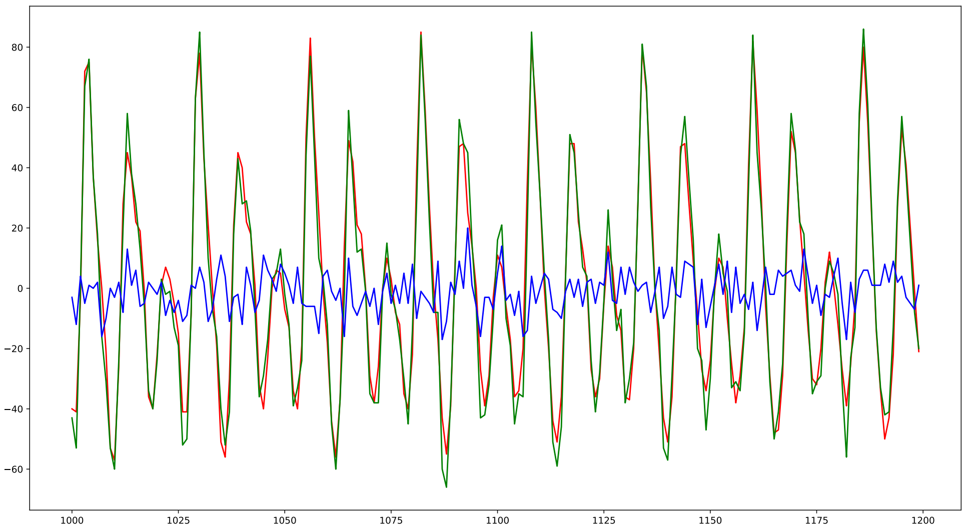


Figure 9.13: Quality of the result: original signal in red, estimated signal in green and blue the difference between the original and the estimated signal.

Moreover, figure 9.13 shows the quality of the estimated signal. Compared to figure 9.9, the difference (plotted in blue) is not completely equal to zero. This means the system is not capable to remove the Gaussian noise from the signal.

Another way to visualize the noise still present in the signal is to analyze the spectrogram of the estimated signal. Figure 9.14 shows the spectrogram. In the spectrogram, dark means no energy in the signal. Red means a bit of energy and the brighter the higher the energy in the signal. One can see that there are a few bright zone along the signal which represent the actual source signal. However, the spectrogram is mainly red due to the noise.

In conclusion, we can say that when Gaussian noise is added to the

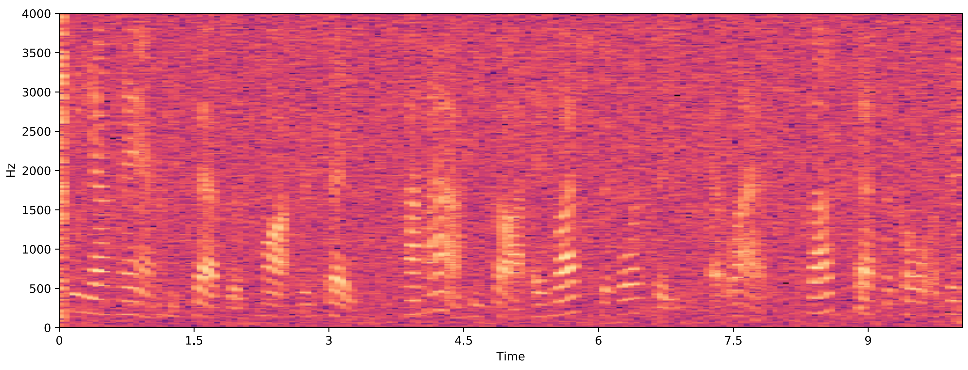


Figure 9.14: Spectrogram of the estimated signal for source 1.

recorded signals, the system still performs well. However, the noise is still present in the estimated signals of the sources.

9.4.6 Separation result using only two microphones

The results presented in the previous sections were using the four microphones presented in our standard setup in section 9.4.2. This setup was chosen due to the setup used for the multi-source localization method described in chapter 8. However, such a setup is called a over-determined setup as we use four microphones to separate two sources.

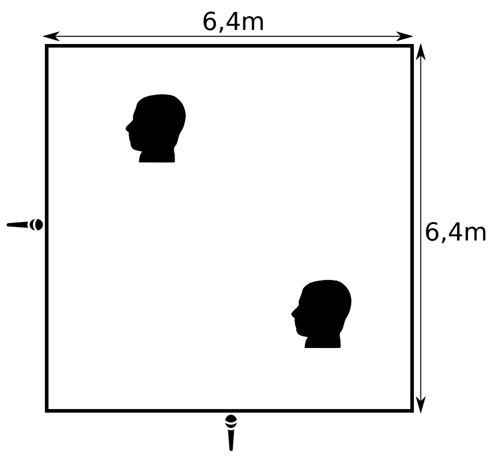


Figure 9.15: Position of the speakers and the two microphones.

In this section, we want to analyze the system performance when using only two microphones. Therefore, the setup has been slightly modified. We only use two microphones each placed in the middle of two adjacent walls of the room. For this experiment, we generate the sound according to the setup illustrated in figure 9.15.

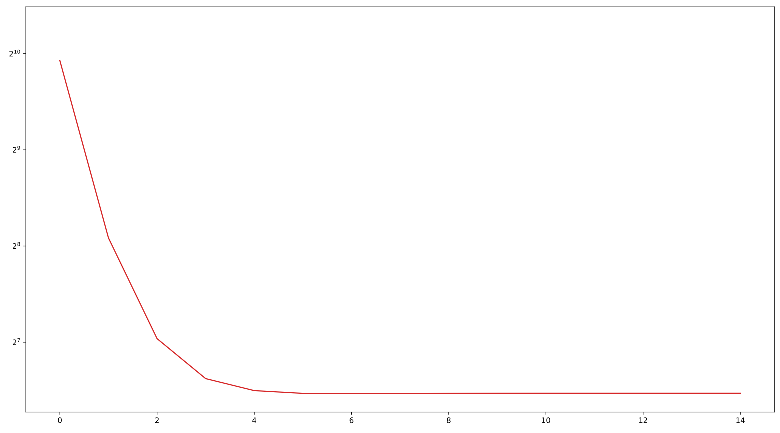


Figure 9.16: Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps.

When running the separation algorithm, we can see that the convergence of the Gibbs algorithm is as fast as with four microphones, as illustrated in figure 9.16.

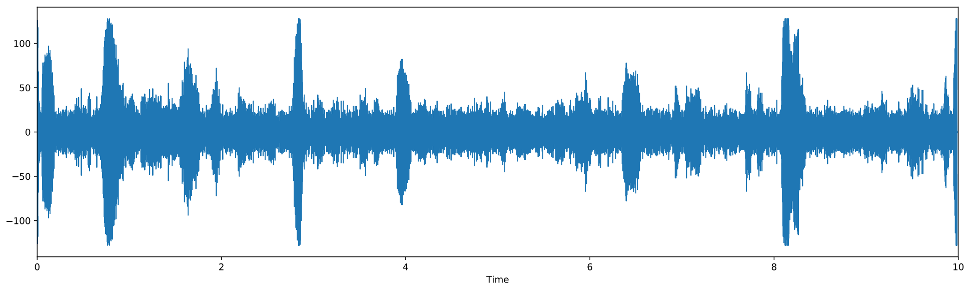


Figure 9.17: Reconstructed signal of source 1 after sweep 6.

The final result is shown in figure 9.17. One can see the signal to be separated. However, as in the previous section, the final result still contains a lot of noise, which has been added to the signals recorded by the microphones as our sound simulator does not simulate reverberations.

9.5 Running the Bayesian sampling machine

Until now all the simulations presented above were performed to validate the algorithm and show the impact of different parameters on the result. How the Bayesian sampling machine was adapted to the sound source separation problem was already described in section 9.3. However, no simulations or experiments implemented the machine. This is the purpose of this section.

In the overall architecture illustrated in figure 9.1, one can recognize the matrix-wise placement of the OP-blocks as it is done in the Bayesian machine used for the sound source localization task. Here, this machine is not used to compute the entire distribution but to draw from the desired distribution. Hence the replacement of the counters at the end of the lines by the draw-gate.

Remember, for each $j \in \{0, ..., J\}$ and for each $t \in \{0, ..., T\}$, we draw from equation 9.11 which is rewritten here:

$$P(S_t^j | \boldsymbol{s}_{\neq t}^{\neq j}, \boldsymbol{c}^\star, \boldsymbol{m}) = \frac{1}{z'} \prod_{i=1}^{i=I} \mathcal{N}(m^i_{t+\delta_i^j}; a(\boldsymbol{c}^{\star j}, i) \cdot s_t^j + \textstyle\sum_{l \neq j} a(\boldsymbol{c}^{\star l}, i) \cdot s^l_{t+\delta_i^j - \delta_i^l}, \sigma^2) \tag{9.13}$$

In this equation, there are mainly four likelihoods to multiply and, as stated above, the likelihoods are following Gaussian distributions. The draw-gate and its specific way of operating, as explained in section 4.3.1, guarantees the Gibbs algorithm to converge. However, while implementing the machine, we looked closer at the Gaussian distributions and noticed that the variance of these normal distributions had a huge impact on the result.

In some cases, the four Gaussian distributions are clearly separated into two groups, as shown in figure 9.18. In this figure, all the g possible values from -128 to +128 are on the X-axis. On the Y-axis, the probability is provided. Naturally, the product of these likelihoods is zero, as illustrated by the purple line in the graph. In this case, the stochastic machine, which computes the maximum of the overlap of these four normal distributions would run indefinitely since there is no overlap. Typically, for such cases the variance needs to be bigger. Moreover, to avoid waiting too long for a potential result of the draw-unit, we set a maximal bit streams size after which the draw of the distribution is stopped and a random number is taken instead. In our simulations, the parameter maxstep was set to 300.

Opposed to the cases mentioned above with no result computed by the draw-gate, we also encounter situations in which the four Gaussian distributions nearly entirely overlap. Note that these cases constitute the majority of the encountered cases, which helps the algorithm to converge

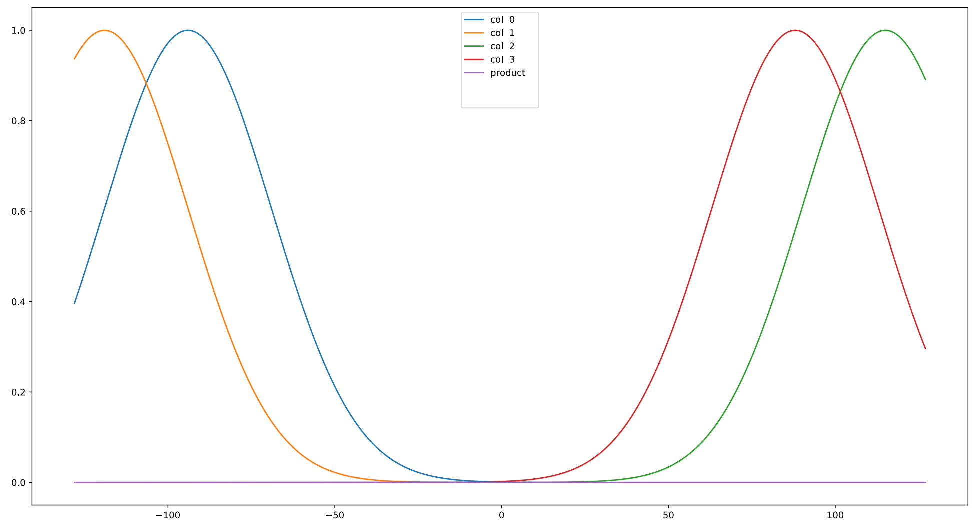


Figure 9.18: The four Gaussian distributions of the likelihoods separated in two main groups.

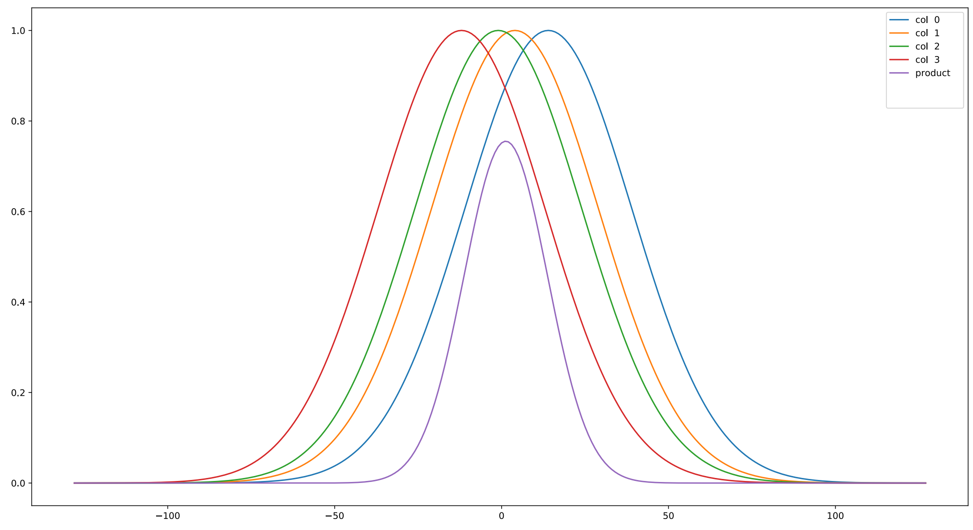


Figure 9.19: The four Gaussian distributions of the likelihoods superposed.

quickly. Such an example is shown in figure 9.19. The product, which is equal to the geometrical overlap of these Gaussian distributions is very high, as shown by the purple line in the graph. In this case, when running the draw-gate, we would have several lines, which very quickly provide a 1 in the stochastic bit stream. Typically, we have the whole intervals (as in this example the lines equivalent to the signal values between -19 and +19) of lines which are at 1. In this case, the draw-gate follows a specific algorithm, explained in section 4.3.1. However, for our application, following the original method of the draw gate would slow down the convergence of the machine as the most probable value (the maximum of the purple line in the graph) would rarely be drawn. Therefore, we replaced the original draw-gate by simply taking the mean value of the min and the max of the obtained interval. In the current example, we obtained the interval $[-19; +19]$ and hence the value 0 would be taken. In the graph, this is the same as taking value on which the Gaussian distribution is centered.

Moreover, we noticed that the variance chosen for the Gaussian distribution of the likelihoods in the equation 9.13 highly depends on the setup in the room. More precisely, the positions of the microphones affect the result. We still have to improve the method to set the variance.

9.5.1 Simulations on the modified Bayesian sampling machine

We simulated the modified Bayesian sampling machine using the standard experimental setup, described in section 9.4.2. The machine computed 20 sweeps of the Gibbs algorithm. The recording time was set to 2 seconds (16000 samples). The BSM was implemented in our simulator in Python using a some C++ code to accelerate the computation. For the 20 sweeps, the simulation takes around 15.05 seconds.

Note that, the machine converged very quickly to a result. The graph in figure 9.20 shows that already after nine sweeps the quality of the obtained result became very good. Moreover, one can observe the stochastic side of the machine as the quality of the result does not stagnate but slightly changes after sweep 10. This is due to the random choices in the machine.

The computed result for source 1 is provided in figure 9.21. In red, the original signal. In red the original signal, in green the estimated signal and in blue the difference of them. Beside the errors at the beginning and the end of the recording interval, the numbers of samples where the blue line (different between original and estimated signal) is not zero is very low.

9.6 Conclusion

In this chapter, a probabilistic method to perform sound source separation has been presented. The probabilistic model has been introduced. To perform

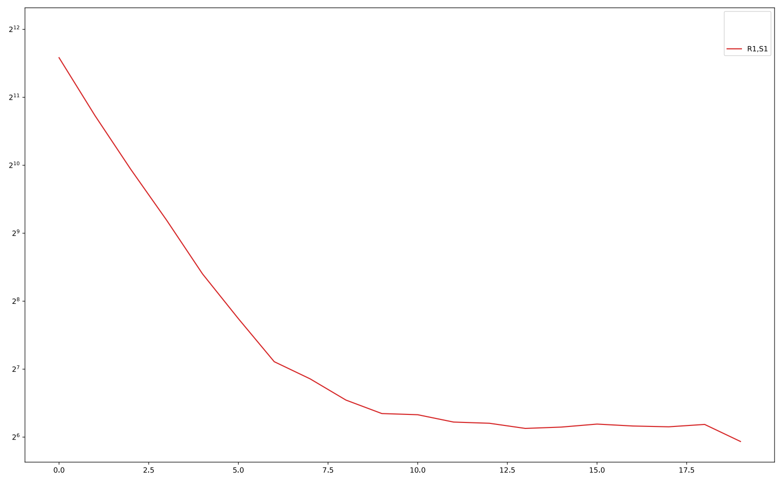


Figure 9.20: Mean squared distance between the original source signal and the reconstructed signal as a function of the number of sweeps.

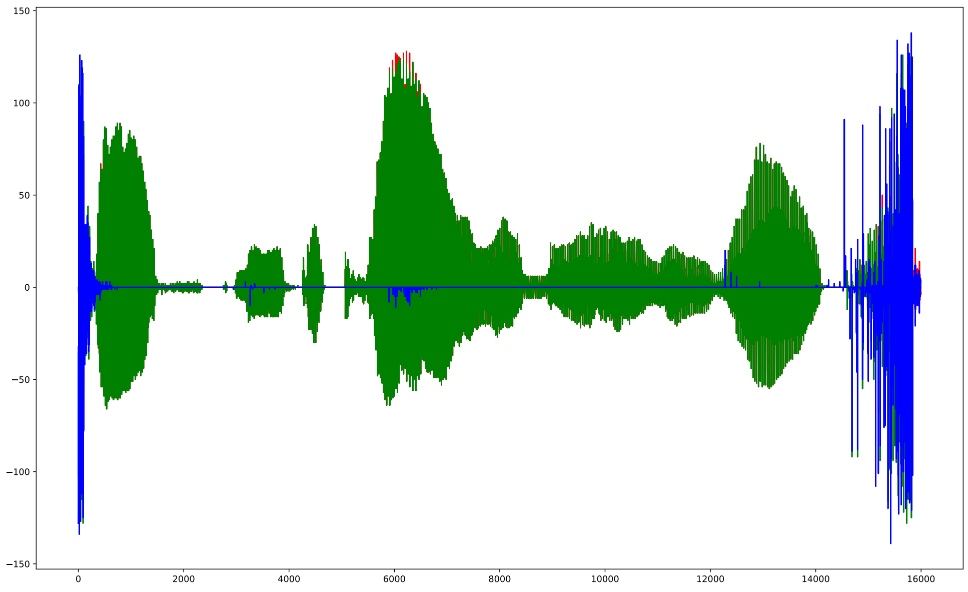


Figure 9.21: Reconstructed signal of source 1 after sweep 20.

approached inference on this model, the Gibbs algorithm has been used to sample the searched space. Therefore, the Bayesian sampling machine, introduced in section 4.3, has been used. Multiple high-level simulations have been presented to show the performance of the algorithm. Finally, the Bayesian sampling machine has been implemented in the simulator and simulations have been run to demonstrate the overall effectiveness of the system.

This method works entirely in the temporal domain. The Gibbs algorithm samples the source signal to estimate it based on the recorded microphone signals.

As mentioned in section 9.2, we focused on a subproblem of the main problem $P(\boldsymbol{C}, \boldsymbol{S}|\boldsymbol{m})$ which aims to use Gibbs to estimate the source positions and source signals at the same time. We did so because when we first tested to run the Gibbs algorithm on $P(\boldsymbol{C}, \boldsymbol{S}|\boldsymbol{m})$, the algorithm was not converging and hence not finding any solution. This is due to the search space which became bigger and especially due to the fact that we would sample two different types of variables. Therefore, we identified two sub-problems, which we worked on:

1. $P(\boldsymbol{C}|\boldsymbol{s}^\star, \boldsymbol{m})$: sampling the position of the sound sources knowing the actual emitted signals $\boldsymbol{s}^\star$ and the signals recorded by the microphones $\boldsymbol{m}$

2. $P(\boldsymbol{S}|\boldsymbol{c}^\star, \boldsymbol{m})$: sampling the signals of the sound sources knowing their actual position $\boldsymbol{c}^\star$ and the signals recorded by the microphones $\boldsymbol{m}$

The first sub-problem is actually very unrealistic as this method needs the source signals $\boldsymbol{s}^\star$ to work. However, we still implemented that method and noticed that the Gibbs algorithm does not converge neither.

$P(\boldsymbol{C}^j|\boldsymbol{c}^{j\neq i}\boldsymbol{s}^\star, \boldsymbol{m})$

So we decided to focus on the second sub-problem for which the Gibbs algorithm converges. Moreover, using the multi-source localization method presented in chapter 8, one can estimate the source positions $\boldsymbol{c}^\star$ and then run the sampling method to estimate the source signals. This is the core of this chapter.

Chapter 10

Conclusion & Discussion

This work tries to address the present need for alternative computing architectures. We proposed two types of machines dedicated to different kinds of applications. The first one, called the *Bayesian machine* (BM), is dedicated to inference problems with small search spaces such as in sensor fusion applications. The second one, which is a more general machine, is called the *Bayesian sampling machine* (BSM). It is devoted to more general inference problems having a large sampling space and is based on the Gibbs sampling method.

We focused on two audio signal processing applications, namely Sound Source Localization (SSL) and source separation. For both tasks, the problem has been modeled using a probabilistic model and a dedicated machine has been designed to perform the Bayesian inference.

We presented the current efforts done by the rebooting computing community in chapter 2. Then, in chapter 3, we defined all the mathematical tools, such as Bayesian programming, required to design our machines. Both our architectures are described in chapter 4, including two simple pedagogical examples to illustrate the different dimensions and parts of each machine. Chapter 5 introduces background knowledge for audio signal processing and defines the signal processing tasks we treated, before providing a short overview of the existing techniques. Chapters 6 to 9 are dedicated to the methods we propose to solve both signal processing applications.

A first approach for SSL is presented in chapter 6. A probabilistic model is defined, which fuses the Inter-channel Phase Difference (IPD) features to estimate the position of the source. The BM was adapted to the current problem. Two optimizations were made to improve the BM. First, we addressed the temporal dilution problem by introducing the slicing technique, which speeds up the computation time by a factor of up to 10^3 and hence reduces the energy required by the machine. Second, we reduced the memory need of the machine since it is large part of the circuit area requiring a lot of power. Therefore we developed an on-chip likelihood computation method. Several simulations were presented to show the impact of the different parameters of the machine on the overall result. The localization performance was tested in simulation and in real world experiments. Furthermore, the circuit was implemented in VHDL to emulate the system on an FPGA. We also performed ASIC simulations to quantify the circuit activity.

As this method uses the IPD, it requires to compute the Fourier transform since the IPD computation needs the time-frequency representation of the signal. Our simulations on FPGA showed that the component corresponding to the Fourier transform and the IPD calculation takes approximately 35% of the entire circuit area. Therefore we developed another method in chapter 7, which works entirely in the temporal domain. A probabilistic model was proposed to run this temporal method on our BM. The on-chip likelihood computation method was modified and simulations showed the potential of this method. However, when dealing with real microphone signals, the quality of the results we obtained is not very satisfying. This is partly due to the probabilistic model, which relies on an attenuation model to localize the source. The attenuation model is a limiting factor to this approach. Therefore we proposed another version, which omits the need of an attenuation model. This method was tested in simulations.

The source separation method proposed in chapter 9 awaits the source positions as input. This motivated the work in chapter 8, which is about a multi-source localization method. This approach uses the temporal mono-source localization method of chapter 7 as a basic block but adds a frame-wise fusion module to achieve multi-source localization. The proposed improvement necessitates to add a simple circuit (an additional counter bank) to the existing circuit of chapter 7. Note that this multi-source localization method was designed with the circuit design in mind. The method was validated in simulations.

Chapter 9 was devoted to the second signal processing application, namely the sound source separation. For this task, the sampling space is very large. For example, to separate 10 seconds of microphone signals and estimate the 2 source signals, we must determine 160000 variables to specify the model, each being discretized on 256 values. Hence, one needs a different inference method since the exact inference method is not suitable for such a large search space. Therefore we used a Gibbs sampler. We defined a probabilistic model for this application and used our second architecture, which is a more generic sampling machine: the *Bayesian sampling machine* (BSM). We adapted the BSM to our source separation problem and ran the machine on our simulator. The simulations showed a great performance in the separation result.

10.1 Discussion

The goal of this project is to promote a non conventional way of computing. We developed two stochastic machines dedicated to Bayesian inference. As they are using stochastic computing, they may be suitable for low-power applications. However, we still need to compare our architectures to low-power microprocessors or other dedicated hardware. For this purpose, we are currently developing an ASIC prototype of the *Bayesian machine*. The first hardware simulations presented in this work showed some encouraging results.

Usually, the results of computer programs are numerical values. Taking into account the uncertainty of the model and of the data is one step further. In this case, we are not any more interested in values but in probability distributions on the possible values. Sampling machines aim at providing Independent and Identically Distributed (IID) samples of these probability distributions. In chapter 9, we proposed a way to sample probability distributions in high dimensional spaces using stochastic computing. In some cases, for example for source separation, the approach works very well. However, for other problems such as the simultaneous separation and localization problem, the method fails to converge. In this case, the *temporal dilution problem* may stop the production of samples revealing regions of low probability density. Here, even conventional machines have numerical problems. Until now, we did not find any general technique to escape from these low density regions. Such general purpose strategy probably does not exist as it is related to the curse of dimensionality and NP problems. We conjecture that making better and more informative models is the only way out in this case.

We hope this work represents a useful and further step in providing an alternate computation scheme to conventional processors. Moreover, we hope it will push probabilistic programming forward by providing dedicated hardware for inference. Finally, this could foster the simultaneous use of models and data.

Appendix A

Notations

In this part the notations used in the different parts of this thesis will be defined. Due to the multidisciplinary nature of the work, some notations differ from the conventional notations used in the specific domain and community.

A.0.1 Specific notations for probabilistic inference

In the area of probabilistic inference we need the following notations:

- M denotes a stochastic variable.
- m denotes a value of the stochastic variable M.
- $\boldsymbol{M}$ denotes an array of stochastic variables.
- $\boldsymbol{m}$ denotes values of $\boldsymbol{M}$.
- $\boldsymbol{M}_k$ denotes k-th line of $\boldsymbol{M}$.
- $\boldsymbol{m}_k$ denotes values of $\boldsymbol{M}_k$
- M_k^j denotes an element of $\boldsymbol{M}$ (note that since it is not multidimensional, the variable is not in bold).
- m_k^j denotes its value (note that since it is not multi-dimensional, the variable is not in bold).
- $\boldsymbol{V}$ denotes a multidimensional vector
- $\boldsymbol{V}_k$ denotes k-th element of a multidimensional vector

- v_k denotes its value (note that since it is not multi-dimensional, the variable is not in bold)
- $\boldsymbol{M}_{\neq k}$ denotes all the elements of a multidimensional matrix except the elements in the k-th line
- $\boldsymbol{M}_{\neq k}^{\neq j}$ denotes all the elements of an array except the variable in the k-th line and j-th column

A.0.2 Specific notations for audio signal processing

- $\mathcal{M}$ is the Fourier transform of a signal m
- $\mathcal{M}_{k,l}^{1}$ is the value of the microphone 1 m in the time-frequency domain for the k-th frequency in the l-th time frame

A.0.3 Specific notations for the applications (SSL and source separation)

- $\boldsymbol{M}$ is the matrix of the stochastic variables associated to the sounds recorded by the microphones.
- $\boldsymbol{m}$ is a particular realization of the conjunction of $\boldsymbol{M}$.
- M_t^i is the stochastic variable associated to microphone i at time t.
- m_t^i is a value of the microphone i at time t.
- $\boldsymbol{S}$ is the matrix of stochastic variables associated to all the sounds emmited by the sources.
- $\boldsymbol{s}$ is a particular realization of the conjunction $\boldsymbol{S}$.
- S_t^j is the stochastic variable associated to the sound emmited by source j at time t.
- s_t^j is a value of the source j at time t.
- $\boldsymbol{C}$ is the matrix of the stochastic variables associated to the coordinates of the different sources.
- $\boldsymbol{c}$ is a particular realization of the conjunction $\boldsymbol{C}$.
- $\boldsymbol{C}^j$ is the tupple of the stochastic variables associated to the coordinates of the source j.

- $\boldsymbol{c}^j$ is a particular realization of the conjunction $\boldsymbol{C}^j$.
- C_x^j and C_y^j are the stochastic variables associated associated to the coordinates in x-axis and y-axis of source j
- c_x^j and c_y^j are possible realization of C_x^j and C_y^j.

Appendix B

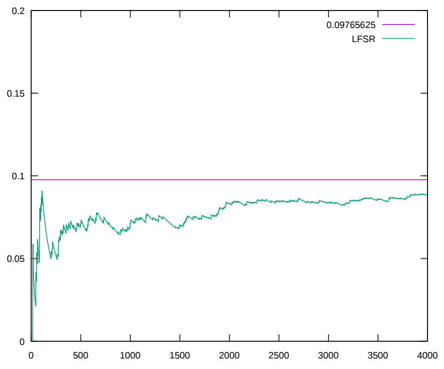
LFSR issue

In this appendix, an LFSR issue is discussed which occurs when using the LFSR to generate the stochastic bit streams. Stochastic bit streams, which are one of the principal components of both our Bayesian machines are generated using RNGs (Random Number Generators). Luckily, the machines do not require random numbers with cryptographic quality. Therefore, more simple mechanisms can be used to generate the random numbers. Currently, we use a 32-bit Galois LFSR in our implementations. The Galois LFSR has a specific period of 2^{32} after which the output repeats. This period is big enough for our applications as we perform various techniques to speed up the computation such as the max-normalization presented above.

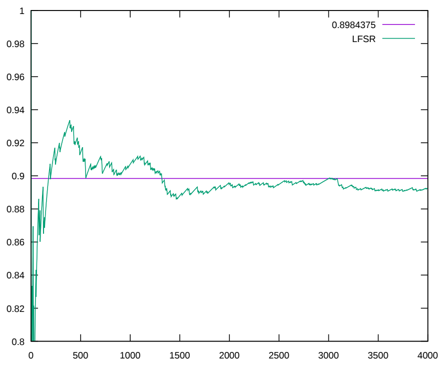
However, when generating the stochastic bit stream with the LFSR as RNG, we noticed that depending on the seeds, we can obtain very bad results, as shown in figure B.1a and figure B.1b. In both figures, the value of the computed stochastic bit stream is provided in green while the reference probability value is given by the purple line.

One can notice the offset present in both lines, even after 4000 steps. Looking at the percentage of the offsets, one can see that the lower probability in figure B.1a is generated with an error of approximately 10%. Fortunately, concerning the bigger probability, the percentage is smaller.

Using the max-normalization technique described in section 4.2.5, we modify the probabilities to obtain higher probabilities for the computation. This luckily leads to probabilities that are better generated by the LFSR, as illustrated in figure B.1b. Hence, the issue described above for lower probabilities is not a big problem. Especially since we are mainly interested in the maximum of the probability distribution for our application. For reproducibility, we used a Galois 32-bit LFSR to obtain these graphs. The



(a) LFSR generating a probability of 0.09765625.



(b) LFSR generating a probability of 0.8984375.

Figure B.1: Example of bit streams generated using and 32 bits LFSR.

chosen seed was 424569598. According to [1], the feedback polynomial is

$$x^{32} + x^{22} + x^2 + 1.$$

[1] `http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf`

Appendix C

Proof draw gate

In this appendix, the mathematical proof for the correctness of the algorithm implemented by the draw gate in the Bayesian sampling machine is provided. This proof was done by Didier Piau as a member of the MicroBayes project.

On the action of the D-door on multiple lines

Did – MicroBayes

June 6, 2019

Abstract

The D-door combines independent i.i.d. bitstreams of known parameters, to produce an output bitstream with some specific one-dimensional asymptotic marginal distribution.

First, following Droulez, we describe the action of the D-door on pairs of bitstreams of parameters p_1 and p_2 and we revisit the existing proof that the D-door then outputs a Markovian bitstream with parameter $p_1/(p_1 + p_2)$. Then we extend this result to n independent bitstreams of given parameters, for every $n \geqslant 2$. Finally, we present some more conjectural remarks.

1 Action of the D-door on two lines

Our first aim is to (re)prove Result 1, due to Droulez [1], using an approach that we will next adapt to encompass the multiline case:

Result 1. *If the input of the D-door is made of two independent lines* 1 *and* 2 *and if each line is i.i.d. Bernoulli, then the output line is a non degenerate Markov chain on* $\{1, 2\}$*, whose stationary distribution is proportional to the parameters of the input lines* 1 *and* 2*.*

1.1 First version of the model

Consider some lines 1 and 2, that produce bitstreams $(B_t^1)_{t\geqslant 0}$ and $(B_t^2)_{t\geqslant 0}$, in a synchronized way, and send them to a door. Each time a pair of bits $B_t = B_t^1 B_t^2$ in $\{0, 1\}^2 = \{00, 01, 10, 11\}$ is sent to the door, the door emits one single signal X_t^o in $\{0, 1, 2\}$, defined as follows.

- If $B_t = 00$ then $X_t^o = 0$.
- If $B_t = 10$ then $X_t^o = 1$.

- If $B_t = 01$ then $X^o_t = 2$.
- If $B_t = 11$ and if the last nonzero signal emitted by the door is 1 then $X^o_t = 2$.
- If $B_t = 11$ and if the last nonzero signal emitted by the door is 2 then $X^o_t = 1$.

We call this mechanism the D-door (for two lines) and we are interested in the properties of the process $(X^o_t)_{t\geqslant 0}$ restricted to the times t such that $X^o_t \neq 0$.

We assume that $(B^1_t)_{t\geqslant 0}$ and $(B^2_t)_{t\geqslant 0}$ are independent sequences of independent and identically distributed bits of respective Bernoulli distributions $\Pr(B^1_t = 1) = p_1$, $\Pr(B^1_t = 0) = q_1$, $\Pr(B^2_t = 1) = p_2$, $\Pr(B^2_t = 0) = q_2$, for some given $(p_1; p_2)$ in $]0, 1[$, where we denote $q_k = 1 - p_k$ for every k.

1.2 Revised model

We now provide a simpler description of the process $(X^o_t)_{t\geqslant 0}$ when $X^o_t \neq 0$. Consider that the pair of lines produces a sequence $(B_t)_{t\geqslant 0}$ of independent and identically distributed symbols 11, 10 and 01, with distribution

$$\Pr(B_t = 11) = p_1p_2s \quad \Pr(B_t = 10) = p_1q_2s \quad \Pr(B_t = 01) = q_1p_2s$$

where s is a normalizing factor, defined as

$$\frac{1}{s} = 1 - q_1q_2 = p_1 + p_2 - p_1p_2 = q_1p_2 + p_1 = p_1q_2 + p_2$$

Then the D-door acts as follows:

- If $B_t = 10$ then $X_t = 1$.
- If $B_t = 01$ then $X_t = 2$.
- If $B_t = 11$ and $X_{t-1} = 1$ then $X_t = 2$.
- If $B_t = 11$ and $X_{t-1} = 2$ then $X_t = 1$.

The process $(X_t)_{t\geqslant 0}$, with values in $\mathbb{Z}_2 = \{1, 2\}$, corresponds to the process $(X^o_t)_{t\geqslant 0}$, with values in $\{0, 1, 2\}$, observed only when $X_t \neq 0$. From now on, we use the stream $(X_t)_{t\geqslant 0}$.

1.3 Proof of Result 1

We first note that the process $(X_t)_{t\geqslant 0}$ is such that $X_t = F(X_{t-1}, B_t)$, for some function $F : \mathbb{Z}_2 \times \{11, 10, 01\} \to \mathbb{Z}_2$ which can be written down explicitly as $F(x, 10) = 1$ and $F(x, 01) = 2$ for every x in $\mathbb{Z}_2$, $F(1, 11) = 2$ and $F(2, 11) = 1$. Since each B_t is independent of $(X_u)_{u\leqslant t-1}$, this representation proves that $(X_t)_{t\geqslant 0}$ is a Markov chain on $\mathbb{Z}_2$.

As for every finite irreducible Markov chain, the transition matrix $p = (p_{x,y})_{x,y}$ of the chain, indexed by $\mathbb{Z}_2 \times \mathbb{Z}_2$ and defined as

$$p_{x,y} = \Pr(X_t = y \mid X_{t-1} = x)$$

for every x and y in $\mathbb{Z}_2$, determines the unique stationary distribution π of the chain, indexed by $\mathbb{Z}_2$, through the fact that π has total mass 1 and satisfies the identity $\pi^t = \pi^t p$, that is, for every y in $\mathbb{Z}_2$,

$$\pi_y = \sum_{x\in\mathbb{Z}_2} \pi_x p_{x,y}$$

In the present case, the transition matrix is

$$p_{1,1} = \Pr(B_t = 10) = p_1 q_2 s \qquad p_{1,2} = \Pr(B_t^2 = 1) = p_2 s$$

and

$$p_{2,1} = \Pr(B_t^1 = 1) = p_1 s \qquad p_{2,2} = \Pr(B_t = 01) = q_1 p_2 s$$

hence

$$\pi_1 = \pi_1 p_1 q_2 s + \pi_2 p_1 s \qquad \pi_2 = \pi_1 p_2 s + \pi_2 q_1 p_2 s$$

Plugging $(\pi_1, \pi_2) = (p_1, p_2)$ in these equations yields true identities hence, the solution such that $\pi_1 + \pi_2 = 1$ being unique, all this shows that

$$\pi_1 = \frac{p_1}{p_1 + p_2} \qquad \pi_2 = \frac{p_2}{p_1 + p_2}$$

as desired.

2 Action of the D-door on multiple lines

Consider now n lines, indexed by k in $\mathbb{Z}_n = \{1, 2, \dots, n\}$, which together produce a stream of n-bits $(B_t)_{t\geqslant 0}$ in a synchronized way, and send them to the D-door.

Each n-tuple of bits $B_t = B_t^1 B_t^2 \cdots B_t^n$ is in $\mathbb{B}_n = \{0, 1\}^n \setminus \{(0, 0, \dots, 0)\}$. Each time some B_t is sent to the D-door, the D-door emits one single signal X_t in $\mathbb{Z}_n$, defined as

$$X_t = \min\{k \geqslant X_{t-1} + 1 \mid B_t^k = 1\}$$

Here, an important convention is that one examines the values B_t^k starting immediately after the position X_{t-1}, in their order of succession for the natural succession order on the discrete circle $\mathbb{Z}_n$ (thus, each $k \neq n$ in $\mathbb{Z}_n$ is followed by $k+1$ and n is followed by 1), and that one stops at the position of the first 1 that one encounters (there is always at least one).

When $n = 2$, one recovers the dynamics described differently previously, now described more compactly.

Compared to the construction when $n = 2$, we omitted the X^o version with values in $\{0, 1, 2, \ldots, n\}$, jumping directly to the X version which is equivalent to observing X^o only when $X_t^o \neq 0$. But again, this new, more general and more convenient, description is strictly equivalent to the model described above when $n = 2$.

Again, the process $(X_t)_{t \geqslant 0}$ is such that $X_t = F_n(X_{t-1}, B_t)$, for some function $F_n : \mathbb{Z}_n \times \mathbb{B}_n \to \mathbb{Z}_n$ which can be written down explicitly.

For example, if $n = 3$, then $F_3(x, 100) = 1$, $F_3(x, 010) = 2$, $F(x, 001) = 3$, $F_3(x, 011) = 2$, $F_3(x, 101) = 3$, $F_3(x, 110) = 2$, and $F_3(x, 111) = x + 1$ for every x in $\mathbb{Z}_3$, with the convention that $3 + 1 = 1$.

The sequence $(B_t)_{t \geqslant 0}$ is i.i.d. and, for every $b = (b_k)_{1 \leqslant k \leqslant n}$ in $\mathbb{B}_n$,

$$\Pr(B_t = b) = s \prod_{k=1}^{n} \left(p_k^{b_k} q_k^{1-b_k} \right)$$

where s is the normalizing factor, defined as

$$\frac{1}{s} = 1 - \prod_{k=1}^{n} q_k$$

Note that each factor $p_k^{b_k} q_k^{1-b_k}$ in the product is simply p_k if $b_k = 1$ and q_k if $b_k = 0$.

Each B_t is independent of $(X_u)_{u \leqslant t-1}$ hence the formula involving F_n proves that $(X_t)_{t \geqslant 0}$ is a Markov chain on $\mathbb{Z}_n$, with transition matrix p and unique stationary distribution π, entirely characterized by the fact that π has total mass 1 .and satisfies the identity $\pi^t = \pi^t p$, that is, for every y in $\mathbb{Z}_n$,

$$\pi_y = \sum_{x \in \mathbb{Z}_n} \pi_x p_{x,y}$$

To compute each $p_{x,y}$, note that a transition from x to y occurs if and only if every site strictly between x and y is at 0 and the site y is at 1, thus,

$$p_{x,y} = s p_y \prod_{x<z<y} q_z$$

where the product enumerates every site starting at $x+1$ and ending at $y-1$, walking along $\mathbb{Z}_n$ along the naturel succession order.

For example, if $n=5$, the product over $3<z<2$ uses the terms $z=4$, $z=5$ and $z=1$. Thus, the transition probabilities starting from 3 are

$$p_{3,1} = sq_4q_5p_1 \; p_{3,2} = sq_4q_5q_1p_2 \; p_{3,3} = sq_4q_5q_1q_2p_3 \; p_{3,4} = sp_4 \; p_{3,5} = sq_4p_5$$

As a consequence, π solves the system

$$\pi_y = s\sum_{x\in\mathbb{Z}_n} \pi_x p_y \prod_{x<z<y} q_z$$

for every y, where, once again, the sums and products use the circular succession order on $\mathbb{Z}_n$. To show that $(\pi_x)_x$ is proportional to $(p_x)_x$, it sremains to check that $(p_x)_x$ solves this system, that is, that, for every y,

$$p_y = s\sum_{x\in\mathbb{Z}_n} p_x p_y \prod_{x<z<y} q_z$$

or, equivalently, dividing by sp_y, that

$$1-\prod_{x\in\mathbb{Z}_n} q_x = \frac{1}{s} = \sum_{x\in\mathbb{Z}_n} p_x \prod_{x<z<y} q_z$$

Using the identity $p_x = 1-q_x$, one sees that the RHS is

$$\sum_{x\in\mathbb{Z}_n}(1-q_x)\prod_{x<z<y} q_z = \sum_{x\in\mathbb{Z}_n}\left(\prod_{x<z<y} q_z - \prod_{x\leqslant z<y} q_z\right)$$

hence the last sum concatenates nicely to get the LHS (recall that the empty product is 1, just like the empty sum would be 0), as desired. Thus:

Result 2. *Result 1 holds for n lines, for every $n\geqslant 2$.*

3 Miscellanea

1. If one uses a new D-door on the output streams of some previous D-doors, even independent, the previous analysis fails. To see why, note that, in this situation, the process $(X_t,B_t)_{t\geqslant 0}$ is again a Markov chain, the process $(B_t)_{t\geqslant 0}$ is now a Markov chain (and not i.i.d.), but $(X_t)_{t\geqslant 0}$ is not a Markov chain anymore since the distribution of B_t depends on B_{t-1} and the distribution of X_t depends on (X_{t-1},B_t) (this is the typical setting of a hidden Markov chain of order 1, often called an M1M1 model). To summarize, $(X_t)_{t\geqslant 0}$ again has a stationary distribution, $(X_t)_{t\geqslant 0}$ again converges to this stationary distribution, but nothing ensures that this stationary distribution depends on the stationary distribution of B_t only through the parameters $\Pr(B_t^1 = 1)$ and $\Pr(B_t^2 = 1)$.

2. In the models solved here, if the probability to observe $B_t = 00\cdots0$ is large, one throws away a large proportion of the input bitstreams. In some related models, ways of reusing the discarded portions of the samples were devised. For example, the well-known von Neumann algorithm producing i.i.d. unbiased bits from i.i.d. biased can be refined to use the 00s and the 11s bits, all discarded in the classical version of the algorithm, to produce more unbiased bits from the same sample. But, first, the range of parameters that are common in the present context might make this venue uninteresting because $00\cdots0$s are rare, and, second, the way to apply such ideas to the present setting is not obvious (to me, at present).

3. The convergence in distribution of the Markovian output stream to its stationary distribution is ruled, at least when not too many steps are involved, by the Perron-Frobenius eigenvalue of the transition matrix of the output. Since any circular order of inlines yields the same stationary distribution, one can wonder whether some particular circular orders would be more effective and if so, what makes them such.

4. More generally, and somewhat more vaguely, there are $(n-1)!$ non isomorphic ways of ordering circularly n input streams, each of these orderings most probably produces a different stationary Markovian measure but with the same marginals. One might try to describe this set of measures, or the convex polytope it generates, or the measure closest to being independent amongst them, to name a few natural questions.

References

[1] Jacques Droulez, *La BM1 à l'intérieur de la BM2*, unpublished, 2016.

Bibliography

[1] Homepage of the bambi project, 2019. URL `https://www.bambi-fet.eu/`. Accessed: 2019-07-07.

[2] Homepage of the ibm q system one, 2019. URL `https://www.research.ibm.com/ibm-q/system-one/`. Accessed: 2019-07-07.

[3] Homepage of the microbayes project funded by labex persyval-lab (anr-11-labx-0025-01), 2019. URL `https://persyval-lab.org/en/sites/content/microbayes`. Accessed: 2019-07-07.

[4] Homepage of the ppaml project, 2019. URL `http://www.darpa.mil/program/probabilistic-programming-for-advancing-machine-Learning`. Accessed: 2019-07-07.

[5] Homepage of the probcomp team at mit, 2019. URL `http://probcomp.csail.mit.edu/`. Accessed: 2019-07-07.

[6] Homepage of the upside project, 2019. URL `http://www.darpa.mil/program/unconventional-processing-of-signals-for-intelligent-data-exploitation`. Accessed: 2019-07-07.

[7] aiCtx Inc. Dynamic neurormorphic asynchronous processor (dynap), 2019. URL `https://aictx.ai/technology/`. Accessed: 2019-07-07.

[8] A. Alaghi and J. P. Hayes. Exploiting correlation in stochastic circuit design. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 39–46. IEEE, 2013.

[9] A. Alaghi and J. P. Hayes. Survey of Stochastic Computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):1–19, may 2013. ISSN 15399087. doi: 10.1145/2465787.2465794. URL `http://dl.acm.org/citation.cfm?doid=2465787.2465794`.

[10] A. Alaghi and J. P. Hayes. Fast and accurate computation using stochastic circuits. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 76:1–76:4, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association. ISBN 978-3-9815370-2-4. URL `http://dl.acm.org/citation.cfm?id=2616606.2616700`.

[11] A. Alaghi, C. Li, and J. P. Hayes. Stochastic circuits for real-time image-processing applications. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2013.

[12] R. Andraka. A survey of cordic algorithms for fpga based computers. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, pages 191–200. ACM, 1998.

[13] S. Araki, H. Sawada, R. Mukai, and S. Makino. Underdetermined blind sparse source separation for arbitrarily arranged multiple sensors. *Signal Process.*, 87(8):1833–1847, Aug. 2007. ISSN 0165-1684. doi: 10.1016/j.sigpro.2007.02.003. URL `http://dx.doi.org/10.1016/j.sigpro.2007.02.003`.

[14] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross. VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing. *IEEE Trans. Very Large Scale Integr. Syst.*, 25 (10):2688–2699, 2017.

[15] B. Barrois, O. Sentieys, and D. Menard. The hidden cost of functional approximation against careful data sizing: A case study. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 181–186, 3001 Leuven, Belgium, Belgium, 2017. European Design and Automation Association. URL `http://dl.acm.org/citation.cfm?id=3130379.3130420`.

[16] B. Behin-Aein, V. Diep, and S. Datta. A building block for hardware belief networks. *Scientific reports*, 6:29893, 2016.

[17] L. Benaroya, F. Bimbot, and R. Gribonval. Audio source separation with a single sensor. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(1):191–199, Jan 2006. ISSN 1558-7916. doi: 10.1109/TSA.2005.854110.

[18] P. Bessière, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha. *Bayesian programming.* CRC Press, 2013.

[19] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

[20] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[21] P. Bofill and M. Zibulevsky. Underdetermined blind source separation using sparse representations. *Signal processing*, 81(11):2353–2362, 2001.

[22] M. Brandstein and D. Ward. *Microphone Arrays: Signal Processing Techniques and Applications*, volume 112. 2001. ISBN 9783642075476. doi: 10.1121/1.1500757. URL `https://www.springer.com/us/book/9783540419532`.

[23] R. K. Budhwani, R. Ragavan, and O. Sentieys. Taking advantage of correlation in stochastic computing. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. doi: 10.1109/ISCAS.2017.8050807.

[24] H. Cai, W. Kang, Y. Wang, L. A. D. B. Naviner, J. Yang, and W. Zhao. High performance mram with spin-transfer-torque and voltage-controlled magnetic anisotropy effects. *Applied Sciences*, 7 (9), 2017. ISSN 2076-3417. doi: 10.3390/app7090929. URL `http://www.mdpi.com/2076-3417/7/9/929`.

[25] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová. Machine learning and the physical sciences. *arXiv preprint arXiv:1903.10563*, 2019.

[26] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76 (1), 2017.

[27] L. N. Chakrapani, P. Korkmaz, B. E. S. Akgul, and K. V. Palem. Probabilistic system-on-a-chip architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):29:1–29:28, May 2008. ISSN 1084-4309. doi: 10.1145/1255456.1255466. URL `http://doi.acm.org/10.1145/1255456.1255466`.

[28] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet. Comparison of self-timed ring and inverter ring oscillators as entropy sources in fpgas. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1325–1330, 2012. doi: 10.1109/DATE.2012.6176697. URL `http://dx.doi.org/10.1109/DATE.2012.6176697`.

[29] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet. A self-timed ring based true random number generator. In *19th IEEE International Symposium on Asynchronous Circuits and Systems, ASYNC 2013, Santa Monica, CA, USA, May 19-22, 2013*, pages 99–106, 2013. doi: 10.1109/ASYNC.2013.15. URL `http://dx.doi.org/10.1109/ASYNC.2013.15`.

[30] K. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet. A very high speed true random number generator with entropy assessment. *Workshop on Cryptographic Hardware and Embedded Sys.*, pages 179–196, 2013.

[31] P. Comon and C. Jutten. *Handbook of Blind Source Separation*. Elsevier Ltd, 2010. ISBN 9780123747266. doi: 10.1016/C2009-0-19334-0.

[32] A. Coninx, P. Bessière, E. Mazer, J. Droulez, R. Laurent, M. A. Aslam, and J. Lobo. Bayesian sensor fusion with fast and low power stochastic circuits. In *Proc. of IEEE Int. Conf. on Rebooting Computing*, 2016.

[33] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*, Phoenix, AZ, USA, 2019. (To Appear).

[34] M. D. Ee292e lecture notes, lecture 5, 2013.

[35] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. Cpu db: Recording microprocessor history. *Commun. ACM*, 55(4): 55–63, Apr. 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133822. URL `http://doi.acm.org/10.1145/2133806.2133822`.

[36] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic

manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018. ISSN 0272-1732. doi: 10.1109/MM.2018.112130359.

[37] A. Deleforge, R. Horaud, Y. Y. Schechner, and L. Girin. Co-localization of audio sources in images using binaural features and locally-linear regression. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 23 (4):718–731, 2015.

[38] A. Dempster, N. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 01 1977.

[39] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974. ISSN 0018-9200. doi: 10.1109/JSSC.1974.1050511.

[40] Y. Dorfan and S. Gannot. Tree-based recursive expectation-maximization algorithm for localization of acoustic sources. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 23(10):1692–1703, 2015.

[41] N. Duong, E. Vincent, and R. Gribonval. Under-determined reverberant audio source separation using a full-rank spatial covariance model. *arXiv e-prints*, art. arXiv:0912.0171, Dec 2009.

[42] H. El-Derhalli, S. L. Beux, and S. Tahar. Stochastic computing with integrated optics. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1355–1360, March 2019. doi: 10.23919/DATE.2019.8714875.

[43] R. R. Fay and A. N. Popper. *Introduction to Sound Source Localization*, pages 1–5. Springer New York, New York, NY, 2005. ISBN 978-0-387-28863-5. doi: 10.1007/0-387-28863-5_1. URL `https://doi.org/10.1007/0-387-28863-5_1`.

[44] C. Fevotte and J. . Cardoso. Maximum likelihood approach for blind audio source separation using time-frequency gaussian source models. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.*, pages 78–81, Oct 2005. doi: 10.1109/ASPAA.2005.1540173.

[45] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982. ISSN 1572-

9575. doi: 10.1007/BF02650179. URL `https://doi.org/10.1007/BF02650179`.

[46] R. Frisch, R. Laurent, M. Faix, L. Girin, L. Fesquet, A. Lux, J. Droulez, P. Bessière, and E. Mazer. A bayesian stochastic machine for sound source localization. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Nov 2017. doi: 10.1109/ICRC.2017.8123681.

[47] R. Frisch, M. Faix, E. Mazer, L. Fesquet, and A. Lux. A cognitive stochastic machine based on Bayesian inference: A behavioral analysis. In *IEEE International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, pages 124–131, 2018. doi: 10.1109/ICCI-CC.2018.8482028.

[48] R. Frisch, M. Faix, J. Droulez, L. Girin, and E. Mazer. Bayesian time-domain multiple sound source localization for a stochastic machine. In *Proceedings of the 27st European Conference on Signal Processing (EUSIPCO'2019)*, 2019.

[49] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, Dec 2013. ISSN 0018-9340. doi: 10.1109/TC.2012.142.

[50] C. Févotte, N. Bertin, and J. Durrieu. Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis. *Neural Computation*, 21(3):793–830, March 2009. ISSN 0899-7667. doi: 10.1162/neco.2008.04-08-771.

[51] B. Gaines. Stochastic computing systems. In *Advances in information systems science*, volume 2, pages 37–172. Springer, 1969.

[52] S. Gannot, E. Vincent, S. Markovich-Golan, and A. Ozerov. A consolidated perspective on multimicrophone speech enhancement and source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(4):692–730, 2017. ISSN 2329-9290. doi: 10.1109/TASLP.2016.2647702.

[53] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue. Timit acoustic phonetic continuous speech corpus. In *Linguistic data consortium*, 1993.

[54] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984. ISSN 0162-8828. doi: 10.1109/TPAMI.1984.4767596.

[55] G. Gimenez, A. Cherkaoui, R. Frisch, and L. Fesquet. Self-timed ring based true random number generator: Threat model and countermeasures. pages 31–38, 07 2017. doi: 10.1109/IVSW.2017.8031541.

[56] P. Gonzalez-Guerrero, X. Guo, and M. Stan. Sc-sd: Towards low power stochastic computing using sigma delta streams. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Nov 2018. doi: 10.1109/ICRC.2018.8638611.

[57] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. *CoRR*, abs/1206.3255, 2012. URL `http://arxiv.org/abs/1206.3255`.

[58] A. Griffin, A. Alexandridis, D. Pavlidi, Y. Mastorakis, and A. Mouchtaris. Localizing multiple audio sources in a wireless acoustic sensor network. *Signal Processing*, 107:54 – 67, 2015. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2014.08.013. URL `http://www.sciencedirect.com/science/article/pii/S0165168414003764`. Special Issue on ad hoc microphone arrays and wireless acoustic sensor networks Special Issue on Fractional Signal Processing and Applications.

[59] J. Grollier, V. Cros, and A. Fert. Synchronization of spin-transfer oscillators driven by stimulated microwave currents. *Physical Review B*, 73(6):060409, 2006.

[60] W. J. Gross, V. C. Gaudet, and A. Milner. Stochastic implementation of ldpc decoders. In *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.*, pages 713–717, Oct 2005. doi: 10.1109/ACSSC.2005.1599845.

[61] E. Habets. Room impulse response generator. Technical report, 01 2006.

[62] O. Häggström. *Finite Markov Chains and Algorithmic Applications.* Cambridge University Press, 2000.

[63] M. Horowitz. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014. doi: 10.1109/ISSCC.2014.6757323.

[64] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley and Sons, 2001.

[65] B. Inc. Akida neuromorphic system-on-chip, 2019. URL `https://www.brainchipinc.com/products/akida-neuromorphic-system-on-chip`. Accessed: 2019-07-07.

[66] E. T. Jaynes. *Probability Theory: the Logic of Science*. Cambridge University Press, 2003.

[67] E. Jonas. *Stochastic Architectures for Probabilistic Computation*. PhD thesis, Massachusetts Institute of Technology, 2014.

[68] E. Jonas. *Stochastic Architectures for Probabilistic Computation*. PhD thesis, Massachusetts Institute of Technology, 2014. URL `http://ericjonas.com/images/pdfs/thesis.pdf`.

[69] E. Jonas and K. P. Kording. Could a neuroscientist understand a microprocessor? *PLOS Computational Biology*, 13(1):1–24, 01 2017. doi: 10.1371/journal.pcbi.1005268. URL `https://doi.org/10.1371/journal.pcbi.1005268`.

[70] H. Kayser and J. Anemüller. A discriminative learning approach to probabilistic acoustic source localization. In *2014 14th International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 99–103, Sep. 2014. doi: 10.1109/IWAENC.2014.6953346.

[71] C. H. Knapp and G. C. Carter. The Generalized Correlation Method for Estimation of Time Delay. *IEEE Trans. Acoust.*, 2:320–327, 1976.

[72] W. Krauth. *Statistical Mechanics : Algorithms and Computations*. Oxford University Press, 2006.

[73] P. Kulkarni, P. Gupta, and M. D. Ercegovac. Trading accuracy for power in a multiplier architecture. *Journal of Low Power Electronics*, 7(4):490–501, 2011.

[74] D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–91, 11 1999. doi: 10.1038/44565.

[75] X. Li, L. Girin, R. Horaud, and S. Gannot. Estimation of the direct-path relative transfer function for supervised sound-source localization. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 24(11):2171–2186, 2016.

[76] X. Li, L. Girin, R. Horaud, and S. Gannot. Multiple-speaker localization based on direct-path features and likelihood maximization with spatial sparsity regularization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(10):1007–2012, 2017.

[77] M. Lin, I. Lebedev, and J. Wawrzynek. High-throughput bayesian computing machine with reconfigurable hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 73–82, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-911-4. doi: 10.1145/1723112.1723127. URL `http://doi.acm.org/10.1145/1723112.1723127`.

[78] A. Lingamneni, C. Enz, J. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011. doi: 10.1109/DATE.2011.5763130.

[79] S. Makino. *Audio Source Separation.* Springer, 2018.

[80] M. I. Mandel, R. J. Weiss, and D. P. Ellis. Model-based expectation-maximization source separation and localization. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 18(2):382–394, 2010.

[81] V. K. Mansinghka, D. Selsam, and Y. Perov. Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv preprint*, arXiv:1404.0099, 2014.

[82] G. Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[83] H. McCabe, S. M. Koziol, G. L. Snider, and E. P. Blair. Tunable, hardware-based quantum random number generation using coupled quantum dots. *arXiv preprint arXiv:1907.00795*, 2019.

[84] D. Menard, G. Caffarena, J. A. Lopez, D. Novo, and O. Sentieys. Fixed-point refinement of digital signal processing systems. In *Digitally Enhanced Mixed Signal Systems*, number Chapter 1, pages 1–37. The Institution of Engineering and Technology, May 2019. URL `https://hal.inria.fr/hal-01941898`.

[85] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, Mar. 2016. ISSN 0360-0300. doi: 10.1145/2893356. URL `http://doi.acm.org/10.1145/2893356`.

[86] D. S. Modha. Introducing a brain-inspired computer: Truenorth's neurons to revolutionize system architecture. *IBM Research*, 2014.

[87] B. Moons and M. Verhelst. Energy-Efficiency and Accuracy of Stochastic Computing Circuits in Emerging Technologies. *IEEE J. Emerg. Sel. Top. circuits Syst.*, 4(4):475–486, 2014.

[88] G. E. Moore et al. Cramming more components onto integrated circuits. *Electronics*, 1965.

[89] J. Mouba and S. Marchand. A Source Localization/Separation/Respatialization System Based on Unsupervised Classification of Interaural Cues. In *Proceedings of the Digital Audio Effects (DAFx06) Conference*, pages 233–238, Canada, Sept. 2006. URL `https://hal.archives-ouvertes.fr/hal-00307889`.

[90] P. Mroszczyk and P. Dudek. The accuracy and scalability of continuous-time bayesian inference in analogue cmos circuits. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1576–1579. IEEE, 2014.

[91] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross. Delayed stochastic decoding of ldpc codes. *IEEE Transactions on Signal Processing*, 59 (11):5617–5626, Nov 2011. ISSN 1053-587X. doi: 10.1109/TSP.2011. 2163630.

[92] F. Nesta, P. Svaizer, and M. Omologo. Robust two-channel TDOA estimation for multiple speaker localization by using recursive ICA and a state coherence transform. In *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, pages 4597–4600, 2009. ISBN 9781424423545. doi: 10.1109/ICASSP.2009.4960654.

[93] F. Neugebauer, I. Polian, and J. P. Hayes. Framework for Quantifying and Managing Accuracy in Stochastic Circuit Design. In *DATE*, pages 3–8, 2017. ISBN 9783981537086.

[94] V. Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 46–98, Princeton Press, 1956.

[95] A. Oppenheim, R. W. Schafer, and C. K. Yuen. Digital signal processing. *Systems, Man and Cybernetics, IEEE Transactions on*, 8:146–146, 03 1978. doi: 10.1109/TSMC.1978.4309917.

[96] A. Ozerov and C. Fevotte. Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):550–563, March 2010. ISSN 1558-7916. doi: 10.1109/TASL.2009.2031510.

[97] A. Ozerov, E. Vincent, and F. Bimbot. A general flexible framework for the handling of prior information in audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(4):1118–1133, May 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2172425.

[98] A. Pedram, S. Richardson, S. Galal, S. Kvatinsky, and M. Horowitz. Dark memory and accelerator-rich system optimization in the dark silicon era. *CoRR*, abs/1602.04183, 2016. URL `http://arxiv.org/abs/1602.04183`.

[99] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11, 2013.

[100] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE transactions on computers*, 60(1):93–105, 2010.

[101] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys. Pushing the Limits of Voltage Over-Scaling for Error-Resilient Applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, Mar. 2017. URL `https://hal.archives-ouvertes.fr/hal-01417665`.

[102] B. Rajendran, A. Sebastian, M. Schmuker, N. Srinivasa, and E. Eleftheriou. Low-power neuromorphic hardware for signal processing applications. *arXiv preprint arXiv:1901.03690*, 2019.

[103] M. Raspaud, H. Viste, and G. Evangelista. Binaural source localization by joint estimation of ILD and ITD. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 18(1):68–77, 2010.

[104] J. Rayleigh. *The Theory of Sound*. Number vol. 1 in The Theory of Sound. Macmillan, 1877. URL `https://books.google.fr/books?id=fj0DAAAAQAAJ`.

[105] S. Rickard, R. Balan, and J. Rosca. Real-time time-frequency based blind source separation. In *in Proc. of International Conference on Independent Component Analysis and Signal Separation (ICA2001*, pages 651–656, 2001.

[106] N. Roman, D. Wang, and G. J. Brown. Speech segregation based on sound localization. *The Journal of the Acoustical Society of America*, 114(4):2236–2252, 2003. doi: 10.1121/1.1610463.

[107] A. Saade, F. Caltagirone, I. Carron, L. Daudet, A. Drémeau, S. Gigan, and F. Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6215–6219. IEEE, 2016.

[108] B. Sanguinetti, A. Martin, H. Zbinden, and N. Gisin. Quantum random number generation on a mobile phone. *Phys. Rev. X*, 4: 031056, Sep 2014. doi: 10.1103/PhysRevX.4.031056. URL `http://link.aps.org/doi/10.1103/PhysRevX.4.031056`.

[109] H. Sawada, R. Mukai, S. Araki, and S. Makino. *Frequency-Domain Blind Source Separation*, pages 299–327. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-27489-6. doi: 10.1007/3-540-27489-8_13. URL `https://doi.org/10.1007/3-540-27489-8_13`.

[110] A.-C. Scherzer and K. Meier. Phase-locking on neuromorphic hardware. 2013.

[111] J. Schnupp, I. Nelken, and A. King. *Auditory Neuroscience: Making Sense of Sound*. 01 2010. ISBN 9780262289757. doi: 10.7551/mitpress/7942.001.0001.

[112] J. M. Shainline. The largest cognitive systems will be optoelectronic. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–10. IEEE, 2018.

[113] H. C. So. *Source Localization: Algorithms and Analysis*, chapter 2, pages 25–66. 2011. ISBN 9781118104750. doi: 10.1002/9781118104750.ch2. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118104750.ch2`.

[114] N. D. Stein. Nonnegative Tensor Factorization for Directional Blind Audio Source Separation. nov 2014. URL `http://arxiv.org/abs/1411.5010`.

[115] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei. Deep probabilistic programming. *arXiv preprint arXiv:1701.03757*, 2017.

[116] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 120:1–120:6, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3520-1. doi: 10.1145/2744769.2751163. URL `http://doi.acm.org/10.1145/2744769.2751163`.

[117] B. Vigoda. *Analog logic: Continuous-Time analog circuits for statistical signal processing.* PhD thesis, Massachusetts Institute of Technology, 2003.

[118] E. Vincent, M. G Jafari, S. Abdallah, M. Plumbley, and M. E Davies. *Probabilistic Modeling Paradigms for Audio Source Separation*, pages 162–185. 01 2011. ISBN 978-1-61520-919-4. doi: 10.4018/978-1-61520-919-4.ch007.

[119] E. Vincent, T. Virtanen, and S. Gannot. *Audio Source Separation and Speech Enhancement.* Wiley Publishing, 1st edition, 2018. ISBN 1119279895, 9781119279891.

[120] D. Wang and G. J. Brown. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications.* Wiley-IEEE Press, 2006. ISBN 0471741094.

[121] D. Wang and J. Chen. Supervised speech separation based on deep learning: An overview. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(10):1702–1726, 2018.

[122] F. Wood, J. W. Meent, and V. Mansinghka. A New Approach to Probabilistic Programming Inference. In S. Kaski and J. Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 1024–1032, Reykjavik, Iceland, 22–25 Apr 2014. PMLR. URL `http://proceedings.mlr.press/v33/wood14.html`.

[123] J. Woodruff and D. Wang. Binaural localization of multiple sources in reverberant and noisy environments. *IEEE/ACM Trans. Audio, Speech, Language Process.*, 20(5):1503–1512, 2012.

[124] Q. Xu, T. Mytkowicz, and N. S. Kim. Approximate computing: A survey. *IEEE Design Test*, 33(1):8–22, Feb 2016. ISSN 2168-2356. doi: 10.1109/MDAT.2015.2505723.

[125] O. Yilmaz and S. Rickard. Blind separation of speech mixtures via time-frequency masking. *IEEE Transactions on Signal Processing*, 52(7): 1830–1847, July 2004. ISSN 1053-587X. doi: 10.1109/TSP.2004.828896.

[126] U. Zoelzer. *DAFX: Digital Audio Effects, 2nd Edition.* Wiley, 2011. ISBN 9780470665992. doi: 10.1002/9781119991298.ch14.Link.

Titre : Machines stochastiques dédiées à l'inférence Bayésienne pour la localisation et séparation de sources

Résumé : L'ordinateur est sans aucun doute l'une des inventions les plus importantes du siècle dernier, dont l'impact ne peut être surestimé. Au fil des années, ils sont devenus de plus en plus puissants grâce à l'optimisation constante des processeurs. Avec un besoin croissant en puissance de calcul, et notamment à cause de l'IA, les processeurs sont devenus plus rapides que jamais. Cependant, à cause des limites physiques, la loi de Moore touche à sa fin. Par conséquent, il est nécessaire de proposer des alternatives. C'est le but de la communauté rebooting computing. Dans ce travail, nous nous proposons d'utiliser le calcul stochastique pour construire des architectures dédiées à l'inférence bayésienne visant une faible consommation d'énergie. Nous avons développé deux machines, à savoir la Bayesian machine (BM) et la Bayesian sampling machine (BSM). Dans cette thèse, nous nous intéresserons à deux applications de traitement du signal : la localisation de sources sonores (SSL) et la séparation de source. Pour la SSL, nous présentons trois méthodes utilisant la Bayesian machine. La première méthode fonctionne dans le domaine temps-fréquence, nécessitant le calcul de la transformée de Fourier. La deuxième est entièrement dans le domaine temporel. La troisième approche est une méthode de localisation multi-sources qui est basée sur la seconde. De plus, nous proposons une technique permettant d'accélérer le calcul stochastique d'un facteur 10^3. Nous avons également développé une méthode de calcul des vraisemblances afin de réduire la mémoire de notre machine. Nous avons simulé les trois méthodes et fait des expérimentations en environnement réel. Nous présentons la consommation d'énergie obtenue via des simulations ASIC. Pour la seconde application, la séparation de source, nous introduisons une machine plus générale, la Bayesian sampling machine, qui est basée sur l'échantillonnage de Gibbs. Nous présentons une méthode basée sur l'échantillonnage pour séparer des sources sonores. Cette méthode a été validée en simulation.

Mots-clés : Machine stochastique, Inférence bayésienne, Localisation de source sonore, Séparation de source sonore

Title: Stochastic machines dedicated to Bayesian inference for source localization and separation

Abstract: Computers are without doubt one of the most important invention of the last century, whose impact cannot be overestimated. Over the years they became powerful, due to the constant optimization of their processors. With the growing need of computing power due to AI, processors have become faster than ever. However, since we are reaching the power wall, Moore's law is coming to an end. Therefore, a young research community called rebooting computing is looking for alternative computation architectures. In this work, we propose to use stochastic computing to build architectures dedicated to Bayesian inference aiming low-power consumption. We develop two machines, namely the Bayesian machine (BM) and the Bayesian sampling machine (BSM). In this thesis, we look at two signal processing applications: Sound Source Localization (SSL) and Source Separation (SS). For SSL, we introduce three methods using the BM. The first one is working in the time-frequency domain and hence uses the Fourier transform. The second one is running entirely in the temporal domain. The third one is a multi-source localization approach based on the previous method. We present a technique to speed up the stochastic computation by a factor of up to 10^3. Moreover, we designed an on-chip likelihoods computation mechanism to reduce the memory needs of our machine. Furthermore, we ran simulations and real-world experiments to validate our methods. We made ASIC simulations to evaluate the power consumption. For the second problem, the source separation, we introduce a more general machine, the Bayesian sampling machine, which is based on the Gibbs sampling approach. We present a sampling method to solve source separation and run simulations to show the effectiveness of this technique.

Keywords: Stochastic machine, Bayesian calculus, Source localization, Source separation

Thèse préparée aux laboratoires / *Thesis prepared at*
LIG (*Laboratoire informatique de Grenoble*),
655 avenue de l'Europe, 38330 Montbonnot et
TIMA (*Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs*),
46 Avenue Félix Viallet, 38031, Grenoble Cedex, France

ISBN: 978-2-11-129260-4