SENSOR NETWORK INFERENCE FROM PARTIAL AND CORRELATED OBSERVATIONS

Submission to Ecole Polytechnique Federale de Lausanne

by

Xavier Contijoch Cullere

Supervised by:

Pascal Frossard

Tamara Tosic

July, 2011

Acknowledgements

"I want to thank Tamara Tosic for the excellent assistance, advice and encouragement along all the different stages of the project. Also to Pascal Frossard to allow me to take part of the LTS4 department, which has been a motivating and enriching experience."

Contents

List of Tables

Chapter 1

Introduction

The target of this work is to study the problem of data reconstruction from partial observations in a Wireless sensor network (WSN) environment. We analyze two different data cases: In the first one, sensor measurements are scalar magnitudes (temperatures). In the second case we extend the model for high dimensional signals (images). When sensor data is scalar, we approach the problem by solving a discrete minimization problem for non-convex functions. Moreover, for multidimensional data we solve a continuous-convex minimization problem by modifying both Total Variation and Matching Pursuit algorithms.

Consider a WSN where links can fail and the structure is varying a lot in function of the application implemented [6].We implement a routing and encoding algorithm robust to arbitrary wireless connection. We also need a distributed data gathering in order to be able to reconstruct the signal from any vertex of the graph and make the encoding and routing process independent from each other. We achieve all this requirements by implementing a gossip algorithm [3].

For the first case, we want to reconstruct the temperature field values by observing partial sensor network measurements. Sensors disseminate their observation in network to one randomly chosen neighbor sensor by the gossip algorithm. After a certain number M of message exchange cycles that is less than the total number of sensors N in the network (under-determined system), we want to guarantee the accurate reconstruction of the network data. The obtained inverse problem based on node observations can be solved if we in addition introduce a priory data knowledge (smoothness). Our data is discretized so we develop an algorithm to minimize a non convex function, which is the hardest computational case due to the slow convergence of the classic numerical algorithms [19], [20] . At first, we propose the basic algorithm, which is not achieving good PSNR results so we develop a Viterbi List algorithm-based to increase the quality of the reconstruction, using different techniques of energy regularization. The results are clearly better,

but for more than a certain number of sensors the computational cost is unaffordable, so we modify the method to avoid the processing of such amount of information. We denote this algorithm Truncated Viterbi List algorithm which basically discards the most irrelevant information in the reconstructions steps.

Moreover, when the dimensions of sensed data is higher we treat the problem from another point of view. We assume that all the sensors participate in the data gathering process, hence we receive observations from all the nodes in the network. Furthermore, each image is compressed before sending it using compressed sensing techniques, therefore data is assumed to be sparse or compressible in some basis. On the decoding side we have M samples of each sensor image, where $M << P$ and P is the image size. To solve the obtained inverse problem we assume a priory data knowledge of the signal (sparsity) and certain correlation amongst neighbor data. We propose two solutions, the first is a joint total variation (JTV) model based on the classical total variation regularization [26] for the minimization of convex-continuous functions. In this case we assume that we know the correlation model between neighbors views. Thus, we use the transforms amongst views within a neighborhood to jointly decode the signals. On the other hand we approach the problem with a Basis Pursuit algorithm based on the $l_1 - l_2$ minimization of a given cost function [21]. We solve it by a joint matching pursuit (JMP) technique, which exploits the correlations amongst the different images to increase the performance of the reconstruction.

Chapter 2

Related work

2.1 Data Gathering in Wireless sensor network

As our framework is a wireless sensor network (WSN), we need a distributed data gathering process for exchange information in an arbitrarily connected network of nodes. The topology of such networks changes continuously, so algorithms under this conditions have to be robust against this changes; in addition, nodes operate under limited energy resources. These constrains motivate the implementation of the gossip algorithm for the WNS data gathering.

In the gossip algorithms, all nodes wants to transmit their message to the rest of the network sensors. The sender, selects a receiver to exchange information and *pulls* its message. After receiving this message this node repeats the same procedure to keep the message running over the network. When all nodes know the original data, the data gathering process finishes.

Gossip algorithms are useful to information exchange and processing in arbitrarily networks. The algorithm is robust to failures in the network nodes and topology changes. All this characteristics make the gossip algorithms practical for WNS.

The first application for the gossip algorithms were the consensus problem [2]. The different nodes in the network have to decide the value of some global parameter exchanging information within a certain neighborhood. For example, if we have temperature sensors, the goal is to know the mean value of temperature within the area covered by the WNS.

In gossip-based protocols, at each time-step, the nodes forward information to one or more random nodes with a certain probability. In function of this probability we can classify the methods into static or adaptive. Pair-wise randomized gossip [3] is a static approach where a random node selects a random neighbor and sends its message to it. Broadcast gossip [4] is basically the same protocol but in each transmission all nodes within a certain area receive the

information of the sender. Geographic gossip [12] combines the geographic routing with the Broadcast gossip, assuming that each node knows exactly its relative position in the network. Thus, they exploit the network topology so the extra cost of communication within multi-hop neighbors is compensated by the number of communications needed to spread the information over all the nodes in the network. Smart gossip [5] is an adaptive method where the probability of each node transmitting is adapted in function of the local topological properties in its surroundings. So, when the topology of the network changes smart gossip allows the distributed adaptation of the data-gathering process.

As we explained before, Gossip algorithms can be applied to solve problem in a WNS in a distributed manner [6], our data gathering process is the same as described in [3].

2.2 The inverse problem

The inverse problem consists in infer the values that characterizes a system with only some measurements of it (less than the number of unknowns). There are many fields where undetermined system must be solved to find an accurate reconstruction of the initial signal. In signal and image processing many studies have been developed on the recovery of initial information from M observations of an N-dimensional signal $M << N$, assuming that the a signal is sparse. We can write the collected observations as:

$$y = Wx,$$

where W is the measurement matrix and x is the original data of dimensions $M \times N$ and $N \times 1$ respectively. In the WSN framework, the recovery of x, can be addressed by a l_0 combinatorial optimization problem, but this is NP-complete and numerically unstable. However, if the measurement matrix W is random and the RIP [1] is guaranteed, then, there are a equivalence between l_1 and l_0, which is more relaxed and stable problem [7]. This works assume that the minimization problem is convex, so data $x \in \mathbb{R}^N$. There exist several linear programing techniques that can recover the original signal [16] [17] under this requirements.

2.3 Compressive sensing

Compressive sensing (CS) [17] [1] is a method which captures compressible signals at a rate significantly below the Nyquist one and recovers it from a small number of linear measurements. If the original signal is $x \in \mathbb{R}^N$, CS states that the N samples of it can be recovered by only $M << N$

linearly projected measurements. In the classical compression algorithms data is first acquired at Nyquist- Shannon sampling rate, and then compressed for efficient storage or transmission. In CS the two processes are combined into a single compressive sampling process, greatly reducing the complexities in data acquisition. The main assumption of CS is the sparsity of data, if we can express a signal x in function of a set of basis $\Psi = [\psi_1, \psi 2, ..\psi_N]$, $x = \Psi s$, like:

$$x = \sum_{i=1}^{N} s_i \psi_i = \sum_{i=1}^{K} s_i \Psi_i, \tag{2.1}$$

where s is the $N \times 1$ column vector of weighting coefficients $s_i =< x, \psi_i >= \psi_i^T x$. The signal is K-sparse if there are only $K(K << N)$ non zero coefficients in s; we expose deeply the sparse properties in Chapter 4. If Φ is a $M \times N$ dimension matrix:

$$y = \Phi x,$$

the reconstruction of the N-dimensional signal is possible if $K \leq \frac{1}{2}M$ [18] solving a combinatorial minimization problem:

$$argmin_s \left\{ \|s\|_{l_0} \quad s.t \quad \Phi\Psi s = y \right\}, \tag{2.2}$$

this is an NP-complete problem and numerically untestable. If we guarantee the incoherence between Φ and Ψ, the restricted isometry property (RIP) [7] of the matrix $\Theta = \Phi\Psi$ is fullfilied and we can solve the problem as using the l_1-minimization technique:

$$argmin_s \left\{ \|s\|_{l_1} \quad s.t \quad \Phi\Psi s = y \right\}. \tag{2.3}$$

An easy way to do it is to construct the measurement matrix as a random matrix whose entries are samples of a Gaussian complying to $\mathbb{N}(1, \frac{1}{M})$. There is some studies [7] [8] that exploit the sparsity of sensor readings by applying compressive sensing theory to sensor data gathering. This is, they collect only $M << N$ measurements, which are linear combinations of sensor readings, as a result the final observations are $Y = \Phi\Psi s$, where $s = \{s_i\}_1^N$. Thus, the computational cost is reduced and and the network lifetime is prolonged. The inconvenience of this algorithms is that the topology of the network is well defined (as a chain or tree distributions). In [13] we can observe an example of distributed algorithm via sparse Random projections.

2.4 Discrete minimization for scalar data setup

In [19] some techniques for inverse problems with discrete data are discussed. They give a general formulation of the problem by extending the continuous data case and by defining a mapping from an infinite-dimensional function space into a finite-dimensional vector space. In this context they can analyze the normal and least-squares solutions. The main problem is the lack of numerically stability, that derives from the fact that in most cases the problem is the projection on a finite-dimensional space of an ill-posed problem. They consider discrete data but they maintain reconstructed solutions in infinite-dimensional spaces and only at the last stage perform a fine discretization of the solution in order to produce a numerical or graphical result. So data have to be quantized twice, first at the emisor side and after the minimization process, at the fusion center to recover data on the discrete domain again. This method can induce problems when minimizing the function in the continuous domain, since the solution can be far removed from the optimum on the discrete one. The solution achieved is closely related to the power of the computer used for the numerical analysis. We avoid this problem by processing the information always in the discrete domain.

2.5 Convex minimization problem for continuous data

The literature of convex optimization has provided a long list of solvers for the l_1-minimization problem in Eq.(2.3), such as Matching Pursuit(MP) [22], least angle regression (LARS) [24] and the absolute shrinkage and selection operator (LASSO) [25]. The first method we propose in our work is based on the constrained minimization of the total variation (TV) [26] of the image. Thus, the equation Eq.(2.3) has to be modified as:

$$argmin_s \left\{ \|s\|_{TV} \quad s.t \quad \Phi\Psi s = y \right\}, \tag{2.4}$$

We will give more details of TV operators and parameters in the following chapters. Our TV method is based on the Chambolle algorithm [27], They propose a fast way to solve different types of inverse problems in image processing like image denoising and zooming.

On the other hand we work with an MP algorithm [22] [23], since the correlation model between the sensor data can be incorporated within it in conveniently. It is an algorithm that iteratively decomposes any function f in a set of redundant functions called atoms. This atoms are chosen in order to best match the signal structures, so MP are procedures to compute adaptive signal representations. We select the appropriate atoms from an extremely redundant dictionary

$\mathbb{D} = g_{\gamma_{\gamma \in \tau}}$, where τ are the indices of the full set of functions generating the dictionary and $||g_\gamma|| = 1$. The vector f can be decomposed into:

$$f = \langle f, g_{\gamma_0} \rangle \, g_{\gamma_0} + \|Rf\|^2 , \tag{2.5}$$

where Rf is the residual vector after approximating f in the direction of g_{γ_0}. We can deduce from Eq.(2.5) that g_{γ_0} is orthogonal to Rf, hence:

$$\|f\|^2 = |\langle f, g_{\gamma_0} \rangle|^2 + \|Rf\|^2 . \tag{2.6}$$

To minimize $\|Rf\|$, we must choose $g_{\gamma_0} \in \mathbb{D}$ such that $|\langle f, g_{\gamma_0} \rangle|$ is maximum. MP applies iteratively the same strategy to the residual component. Using this principles we propose an implementation of an algorithm for a multi view setup, exploiting the correlations amongst images within a sensor neighborhood.

Chapter 3

Framework model

3.1 Sensor network model

We can define a Graph with $G = (V; E)$ where V is a finite set of numbered vertices (sensors) and E represents the edges of the graph, or in other words different paths which connects one vertex to the other. A self-loop is an edge which starts and ends at the same vertex. A path is a sequence of vertices $(v_1, v_2, ..., v_m)$ such that $[v_{i-1}, v_i]$ is an edge for all $1 < i < m$ and a graph is connected when there is a path between any two vertices. Further, a graph is undirected when the set of edges is symmetric, i.e., for each edge $[u, v] \in E$ we also have $[v, u] \in E$. In this work, the graphs are always assumed to be connected, undirected, and have no self-loops or multiple edges. We assume that a network topology (sensor positions) is fixed and known, and the total number of sensors is N. Connections $e(v_i, v_j) \in E$ amongst sensors $v_i, v_j \in V$, where $i, j \in [1; ...; N]$ are nonzero values if sensors are located closer then a predefined threshold and zero otherwise.The threshold value is set to be proportional to the distance amongst sensors.

3.2 Routing algorithm and signal model for scalar sensor measurements

The constraints for the Wireless sensor networks have motivated the design of gossip algorithms. In this work we implement the simplest version of this algorithm.

First of all, the devices capture the real initial sensed values x_k for $k = 1; ...; N$, and request the data from one of they neighbor nodes uniformly at random. The senders are nodes which answer this request, making an aleatory choice from the connection matrix, which contains the

direct paths between all vertices in a matrix format. Senders calculate aleatory values, which can be represented as a matrix A filled with the random values of sensor s that wants to transmit to r at time step t, this is $a_{sr}^{(t)}$, therefore this matrix can be viewed as an identifier of each transmitting node. Then, senders multiply $a_{sr}^{(t)}$ by the observation of the previous time step $y_s^{(t-1)}$. After, they send the result to the arbitrary destinations chose. The receiver makes the addition between this value and his initial sensed value x_r multiplied by his own aleatory value $a_{rr}^{(t)}$, and this will be its current observation for the time step t, $y_r^{(t)}$. The observation for a concrete sensor is:

$$y_r = \sum_{t=1}^{M} x_r a_{rr}^{(t)} + y_s^{(t-1)} a_{sr}^{(t)} p_{sr}^{(t)}, \tag{3.1}$$

where M is the number of iterations, this is the number of cycles (time steps), done before the fusion center collects the data ($M << N$).

We want to rewrite Eq.(3.1) as a function of the initial data x_k for $k = 1; ...; N$ and a set of weights W that represent the M linear combinations amongst data available in the fusion side. The $[M \times N]$ dimensional matrix W, is obtained from the random values used to compute the observations along the different iterations. The random values are sent like a header information in each time step. We can write full observations Y, $[N \times 1]$ system in function of X, $[N \times 1]$

$$Y = WX, \tag{3.2}$$

where ϵ is noise.

We model the energy differences between sensors with a Markov Random Field (MRF). MRF for an undirected graph is a graphical model in which a set of random variables have a Markov property. In the 2-D setting, assume that $S = [1; ...; N] \times [1; ...; N]$ is the set of points called sites. We refer to neighborhood of a fixed s like $\mathbb{N}(s)$, meaning that for the site (k, l) $\mathbb{N}(k, l) = (k + 1, l), (k - 1, l), (k, l + 1), (k, l - 1)$. A random field X if said to be an MRF on S with respect to a neighborhood system $\mathbb{N}$ if and only if: $P(x) \geq 0 \quad \forall x \in X$ and $P(x_i|x_{S-i}) = P(x_i|x_{\mathbb{N}i}) \quad \forall i \in S$. According to the Hammersley-Clifford theorem [28], an MRF can equivalently be characterized by a Gibbs distribution:

$$P(x_i) = \frac{1}{Z} \prod_{i=1}^{N} e^{-\Psi(x_i)}, \tag{3.3}$$

where $Z = \sum_{i=1}^{N} e^{-\Psi(x_i)}$ is the normalization term called partition function and Ψ is the potential or energy function. A clique $c \in C$ is defined as a subset of sites in S in which every pair of distinct sites are neighbors (2-clique). Ψ is a sum of clique potentials over all possible cliques, $\Psi = \sum_{c \in C} V_c(x_i)$. We can approximate our potential function model using 2-clique notation as

$$\Psi(x_i) = \sum_{i \in S} \sum_{j \in \mathbb{N}(i)} (x_i - x_j)^2 , \tag{3.4}$$

replacing Eq.(3.4) into Eq.(3.3) we can write:

$$P(x) = \frac{1}{Z} e^{-\sum_{i \in S} \sum_{j \in \mathbb{N}(i)} (x_i - x_j)^2}. \tag{3.5}$$

Our model data x_i, $i \in [1, ..., S]$ are samples of a Gaussian to satisfy the MRF properties explained in this section

3.3 Routing algorithm and signal model for high dimensional sensor measurements

Basically, the routing algorithm protocol is the same as the described in the last section for scalar data. However, for high dimensional data additional processing is performed prior to the data gathering process, described in Eq.(3.1). As our signal is sparse we use this knowledge to reduce the computational burden of the algorithm. We define the image of sensor i with X_i, of dimension $P = Size \times Size$, $X_i \in \mathbb{R}^P$.We treat a high dimensional matrix data by vectorizing it into a $P \times 1$ vector. Due to its sparsity, $M << P$ coefficients in the appropriate basis are sufficient to exactly recover the signal X_i. Therefore, sensors compute M inner products between X_i and a collection of vectors ${\Phi_j}_{j=1}^M$ as in $z_j =< X_i, \Phi_j >$. We can write the measurements of each sensors like:

$$\begin{pmatrix} z_{11} & \dots & z_{1S} \\ \vdots & \ddots & \vdots \\ z_{M1} & \dots & z_{MS} \end{pmatrix} = \begin{pmatrix} \phi_{11} & \dots & \phi_{1P} \\ \vdots & \ddots & \vdots \\ \phi_{M1} & \dots & \phi_{MP} \end{pmatrix} \begin{pmatrix} x_{11} & \dots & x_{1S} \\ \vdots & \ddots & \vdots \\ x_{P1} & \dots & x_{PS} \end{pmatrix},$$

where S is the number of sensors. We can define the compressed data obtained during the measurement process as $Z_i^T = [z_{1i}, ..., z_{Mi}]$, which are the columns of the matrix Z. Analogously, initial sensor data is given with $X_i^T = [x_{1i}, ..., x_{Pi}]$. We set $\Phi_j^T = [\phi_{1j}, ...\phi_{Mj}]$ to be the columns of Φ so we can write the analytic expression for the compressed signal in each sensor as:

$$Z_i = \sum_{j=1}^{P} x_{ji} \Phi_j \quad i = [1, ..., S], \tag{3.6}$$

where the observation Z_i belongs to $\mathbb{R}^M$ after the compressing step in each sensor. Thus the signal dimensions have been reduced from P to M before being sent through the network.

To propagate the sensors compressed data Z_i we use again the gossip algorithm proposed in Eq.(3.1). As a result, the current measurement of sensor i combined with the neighbor information forms the observation sent to the fusion center: $Y_i = w_i Z_i + w_j Y_j$. After a certain number of message exchange cycles, the network data gathering process can be described as:

$$\begin{pmatrix} y_{11} & \dots & y_{1M} \\ \vdots & \ddots & \vdots \\ y_{M'1} & \dots & y_{M'M} \end{pmatrix} = \begin{pmatrix} \omega_{11} & \dots & \omega_{1S} \\ \vdots & \ddots & \vdots \\ \omega_{M'1} & \dots & \omega_{M'S} \end{pmatrix} \begin{pmatrix} z_{11} & \dots & z_{1M} \\ \vdots & \ddots & \vdots \\ z_{S1} & \dots & z_{SM} \end{pmatrix},$$

$$Y = WZ^T = WX^T\Phi^T.$$

Where $Y = [M' \times M]$ are the final observations, $W = [M' \times S]$ is the network measurement matrix and $Z = [S \times M]$ are the compressed signals representations. If the number of observations M' is lower than the number of sensors, we obtain a matrix W which represents the network compression stage. In this work we assume that W is full rank or what is the same we avoid the network compression stage for high dimensional data.

Note that the compressed image representation of the sensor i and the observation sent to the fusion center (FC) are represented by the rows of Z, $Z_i = [z_{i1}, ..., z_{iM}]$ and Y, $Y_i = [y_{i1}, ..., y_{iM}]$, respectively. We can rewrite the received data in the FC as:

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_S \end{pmatrix} = \begin{pmatrix} \omega_{11} & \dots & \omega_{1S} \\ \vdots & \ddots & \vdots \\ \omega_{S1} & \dots & \omega_{SS} \end{pmatrix} \begin{pmatrix} Z_1 \\ \vdots \\ Z_S \end{pmatrix}$$

Finally we have obtained the same expressions for the measurements in the FC than in the unidimensional case, but each value is now a M dimensional vector. With this notation we can state the observation of sensor i as:

$$Y_i = \sum_{i=1}^{S} \omega_{ij} Z_i. \tag{3.7}$$

For high dimensional data our a priory knowledge is the sparsity of data. In our concrete framework we built a simple set of sensor images composed by three different geometric shapes: a square, a triangle and a sphere. Therefore, we can represent our signal as in Eq.(2.1) with $K = 3$

if we construct our basis functions shifting this geometric figures along the image size range. This property of the data allows us to use Matching pursuit-based algorithms and total variation regulation for smoothing the noise and preserve the edges. On the other hand, to generate all the views from the network we can define some global lineal transforms amongst images that allow us to model the shifts between sensors data. We set ψ_{im} and ψ_{jm} for $m = [1, S]$ to be the same basis of $S \times 1$ vectors but with different labels to distinguish the data generating functions of sensor sensor i and j respectively. In the same way we represent the coefficients of sensors i and j like s_{im} and s_{jm} for $m = [1, S]$, then we can write:

$$x_i = \sum_{m=1}^{N} s_{im}\psi_{im}$$

$$x_j = \sum_{m=1}^{N} s_{jm}\psi_{jm} = \sum_{m=1}^{N} s_{jm}F_{ij}(\psi_{im}).$$

Note that F_{ij} is the same for all the ψ_{im}, $\forall m$, so is a global transform that affects in the same way to all the image basis vectors. To work with high dimensional signals we assume that in all the views we can observe the same image but with different translations, which are modeled by the transforms $F_{ij}(\cdot)$. When we work with the Joint Total Variation algorithm we assume that we know this linear transforms between images $F_{ij}(\cdot)$. Otherwise, for JMP we do not assume the knowledge of the transforms amongst views, but we can estimate them with our algorithm. After the approximation, we can model them as $\widetilde{F}_{ji} = F_{ji} + \epsilon$ where ϵ denotes the uncertainty.

Chapter 4

Problem formulation

4.1 Reconstruction of discrete scalar sensor network readings

In the temperature sensor network, we assume that sensor capture measurements that lay within a set of integer values, and we solve the associated discrete minimization problem. We avoid the algorithms for the minimization of convex functions because this implies the interpolation of the signal to obtain a continuous curve, then find the optimal solution in this domain and finally quantize this value to make the solution fit with the original data. To convert a discrete signal into a continuous one is always problematic, because if the interpolation is not accurately done we can minimize a curve which is not representing the real case.

Our routing and encoding techniques allows a extremely flexible choice of the fusion center (FC), which collects data from the network, so we assume that every sensor needs to reconstruct the network values. Underdetermined set of observations Y and set of weights W Eq.(3.2) are available on the fusion side. We want to find a solution for the discreet set of sensor readings X with $M << N$ measurements:

$$X^* = argmin_X \, ||Y - WX||_2^2 + \lambda \mathcal{R}(f(x)). \tag{4.1}$$

Using the observations Y and the measurement matrix W we have to be able to approximate the best solution for the initial N signal X. We solve this problem under the assumption that data is smooth. This constraint is represented with the term $\mathcal{R}(f(x))$, where λ is a constant. This second part of the optimization problem is the regularization term, which increases the energy

calculation for values that diverge from values of its neighbors, in other words it penalizes data which is not smooth. We compare three types of regularization, two of them are non-weighted and the last one is weighted. For the first two cases, data is compared with the neighborhood of the sensor we are analyzing. They give the same importance to all of them, instead, in the weighted type more relevance is given to vertex which are closer to the sensor that we are trying to approximate. We start defining the non-weighted methods, followed by the weighted one.

4.1.1 Absolute value regularization

The Absolute value (ABS) method for regularization is defined as

$$X^* = argmin_X \left|\left|Y - WX\right|\right|_2^2 + \lambda \sum_{i=1}^{N} \sum_{j \in N(v_i)} \left|\left|x_i - x_j\right|\right|_1, \tag{4.2}$$

where $j \in N(v_i)$ denotes the set of vertices adjacent to v_i. For each sensor the algorithm computes the l_1 norm of the difference between its and its neighborhood's.

4.1.2 Total Variation regularization

Total Variation (TV) is a bit different technique of regularization. Most of the methods only assume that data is smooth, but total variation is also able to detect discontinuities on the set of data [14]. Thus, TV preserves the 'edges' of the signal and at the same time smooths the data within this edges. The regularization term can compute now the discontinuity or the total variation of the signal. We define the gradient operators as in [15], the degree function is given with:

$$d(i) = \sum_{j \in N(v_i)} c(i,j), \tag{4.3}$$

and the graph gradient is given by:

$$(\nabla\varphi)([i,j]) = \sqrt{\frac{c([i,j])}{d(j)}}\varphi(j) - \sqrt{\frac{c([i,j])}{d(i)}}\varphi(i), \quad for\ all\ [i,j] \in E. \tag{4.4}$$

We can also define the gradient norm on the vertex j as:

$$\nabla\varphi_j = \sqrt{\sum_{i \in N(v_j)} \left(\nabla\varphi\right)^2 ([i,j])}, \tag{4.5}$$

Thus, the global equation that has to be minimized is given with:

$$X^* = argmin_X \left|\left|Y - WX\right|\right|_2^2 + \lambda \sum_{i=1}^{N} \sum_{j \in N(v_i)} \left|\left|\left(\nabla\varphi\right)([i,j])\right|\right|_2 \tag{4.6}$$

4.1.3 Weighted distance regularization

Weighted distance (WD) is a particular case of the absolute one, which distributes the energy of each estimation depending on the location of the neighborhood vertex. The influence of the neighbors decays with distance. The global equation that we have to minimize is given with:

$$X^* = argmin_X \, ||Y - WX||_2^2 + \lambda \sum_{i=1}^{N} \sum_{j \in N(v_i)} \left|\left| x_i - \frac{1}{d_{ij}} x_j \right|\right|_1, \tag{4.7}$$

where d_{ij} is the distance between vertex i and j.

4.2 Reconstruction of High dimensional sensor network readings

In this case, the inverse problem is due to the compression at the encoding side, since we assume that the network measurement matrix W is full rank. Using the notation defined in 3.3 we can write:

$$\widetilde{X} = argmin_X \left|\left|Y - WX^T\Phi^T\right|\right|_2^2 + \lambda \mathcal{R}(f(x)). \tag{4.8}$$

As W is a full rank matrix, we can assume that the linear projection ΦX is the resultant inverse problem we have to solve. We address the problem from two point of views: the joint TV minimization problem and the joint MP minimization problem. The sets of multi-view images that capture information from different viewpoints are typically related by geometric constraints, so we exploit this correlation for joint sparse approximation of multi-view images. We consider that the neighborhood of a concrete sensor i accomplishes the previous requirements of correlation, because the sensors in the neighborhood area are spatially close, so the geometric relations between them are strong enough. If $F_{ji}(\cdot)$ is the linear transform between signal j to i, the approximation of sensor i is given by:

$$\widetilde{x_i} = argmin_x \left\{ \left|\left|y_i - x_i^T\Phi^T\right|\right|_2^2 + \lambda \sum_{j \in N_i} (y_i - F_{ji}(x_j)) \right\}. \tag{4.9}$$

Note that if W has a full rank we can consider that the observations y_i in Eq.(4.9) are obtained as $y_i = W^{-1}Wx_i^T\Phi^T$. As a result, we eliminate W from the minimization problem.

4.2.1 Joint Total Variation

Consider the single total variation (STV) denoising model (this is for one single image), it solves:

$$\widetilde{x_i} = argmin_x \left\{ \frac{1}{2} ||y_i - x_i||_2^2 + \lambda J(x), \right\}, \tag{4.10}$$

where y_i is the noisy image from sensor i represented as 2-dimensional matrix of size $P \times P$ and $J(x_i) = \sum_{1 \geq m,n \leq P} |(\nabla_{x_i})_{m,n}|$. We define the discrete gradient operator as in [27], if $u_i \in X$, ∇u is a vector in $Y = X \times X$. Then: $(\nabla u_i)_{m,n} = ((\nabla u_i)^1_{m,n}, (\nabla u_i)^2_{m,n})$

$$(\nabla u_i)^1_{m,n} = \begin{cases} (u_i)_{m+1,n} - (u_i)_{m,n} & \text{if } m < P \\ 0 & \text{if } m = P \end{cases}$$

$$(\nabla u_i)^2_{m,n} = \begin{cases} (u_i)_{m,n+1} - (u_i)_{m,n} & \text{if } n < P \\ 0 & \text{if } n = P, \end{cases}$$

for $m, n = [1; ...; P]$. The idea is to replace the optimization of the image x_i by the optimization of a vector field G that is related to x_i by $x_i = y - div(G)$ and $G = [G_x, G_y] \in \mathbb{R}^2$. The vector field is the one that minimizes:

$$\widetilde{G} = argmin_G \left\{ ||y_i - \lambda div(G)||_2 \right\}. \tag{4.11}$$

We introduce a discrete divergence operator $div : Y \to X$, therefore $div : \mathbb{R}^2 \to \mathbb{R}$ with:

$$(div(G))_{m,n} = \begin{cases} G^1_{m,n} - G^1_{m-1,n} & \text{if } 1 < m < P \\ G^1_{m,n} & \text{if } m = 1 \\ -G^1_{m-1,n} & \text{if } m = P \end{cases}$$

$$+ \begin{cases} G^2_{m,n} - G^2_{m,n-1} & \text{if } 1 < n < P \\ G^2_{m,n} & \text{if } n = 1 \\ -G^2_{m,n-1} & \text{if } n = P. \end{cases}$$

To adapt the problem to our framework, this is, to take advantage of the correlations amongst neighborhoods and implement a joint total variation (JTV) model we redefine the STV formulation

in Eq.(4.10):

$$\widetilde{x_i} = argmin_x \left\{ \frac{1}{2} ||y_i - x_i||_2^2 + \lambda \left[\frac{\alpha J(x_i) + \beta J'(x_i)}{\alpha + \beta} \right] \right\} \tag{4.12}$$

Where $J'(x_i) = \sum_{1 \geq m,n \leq P} \left| (\nabla'_{x_i})_{m,n} \right|$. Thus, we introduce the discrete gradient amongst neighborhood images as:

$$(\nabla' u_i)^1_{m,n} = \begin{cases} \frac{1}{M} \sum_{j \in \mathbb{N}_i} F_{ji} \left[(u_j)_{m+1,n} \right] - (u_i)_{m,n} & \text{if } m < P, \\ 0 & \text{if } m = P, \end{cases}$$

$$(\nabla' u_i)^2_{m,n} = \begin{cases} \frac{1}{M} \sum_{j \in \mathbb{N}_i} F_{ji} \left[(u_j)_{m,n+1} \right] - (u_i)_{m,n} & \text{if } n < P, \\ 0 & \text{if } n = P, \end{cases} \tag{4.13}$$

where M is the sensor i neighborhood size. At this point, we proceed as in the STV case, optimizing the vector field G like in Eq.(4.11). We illustrate deeply the impact of the cross gradient $(\nabla' u_i)$ instead of (∇u_i) in the optimization algorithm in the following chapter.

4.2.2 Joint Matching Pursuit

We propose a joint MP model (JMP) in order to exploit the geometric constraints of a multi view scene. However, JMP improves the PSNR over the classical single MP (SMP). We maximize the following equation:

$$\widetilde{g_{\gamma_m}}, \widetilde{F_i^m} = argmax_{g_{\gamma_m}, F_i^m} \left\{ \sum_{i=1}^{N} \left[\langle R^m x_i, F_i(g_\gamma) \rangle + \lambda \sum_{j \in \mathbb{N}_i} \widetilde{F}_{ji}(\langle R^m x_j, g_\gamma \rangle) \right] \right\}, \tag{4.14}$$

where $R^m x_i$ is the residual signal of sensor i left after subtracting results of previous $m-1$ iterations. Moreover, g_{γ_m} is the atom that best fits the residual signal of sensor i on the iteration m and $F_i^m()$ is the linear transform of this atom. As we will see in the following chapter we can estimate $\widetilde{F}_{ji}(\cdot)$ using some data prior knowledge. After finding the M atoms that maximize Eq.(4.14) and their transforms we can compute the signal reconstruction as:

$$\widetilde{x_i} = \sum_{m=1}^{M} \langle R^m x_i, F_i^m(g_{\gamma_m}) \rangle \, g_{\gamma_m}.$$

Chapter 5

Proposed solutions

5.1 Discrete minimization algorithm for scalar sensor data

5.1.1 Basic reconstruction Algorithm

To decode the signal, value Basic reconstruction Algorithm (BR) suppose that a discrete set of possible values is known. Obviously the range of this set must be large enough to include all the possible values of the original data. We assume that we posses previous knowledge of the signal, so is possible to estimate a certain range of data values $v \in T$ where T is the discrete set. The algorithm is given in several stages:

1. Initialize the values of sensors $x \in \mathbb{R}^N$ to 0. Note that x is the minimization function domain in Eq.(4.1).

2. Compute the energy of x given by Eq.(4.1) varying the value of sensor n, this is $x[n]$ in the range of data set candidates $v \in T$, $f : x \to e$, where $e \in \mathbb{R}$ are the set of energies.

3. Find the minimum energy value for all $e \in R$, and store the value $v \in T$ associated to this energy. We fix this value in $x[n]$, this is the approximation of sensor n in the iteration t. BR follows the following steps:

   * Repeat the step 2 and 3 for all $n = [1, .., N]$ fixing the values of the previous sensors obtained in the second step $x[k]$ $where$ $k = [1, .., n-1]$.

   The process is done for the N sensors, the first approximation of the network values is $x^*(t)$ where t is the iteration number.

4. Compute the global energy of the approximation $x^*(t)$.

5. If $\left|x^{*(t)} - x^{*(t-1)}\right| < \delta$, the minimization problem ends and the best approximation is $x^{*(t)}$. If not, the process is repeated from 2 but starting from the solution found in the current iteration, $x = x^{*(t)}$

This is the simplest algorithm, but it converges too fast and the total combinations of the discrete set $v \in T$ is not large enough. We try to solve this problem introducing a variant on the step 3.

5.1.2 Viterbi list algorithm

The Viterbi List algorithm (VLA) is similar to the basic one, but in the third step instead of saving only the lower energy case, it saves the S lowest energy values. If we want to store two values for each case, we propose to implement the $2Best$ Viterbi algorithm. Note that the computational cost is increasing exponentially as $(SBest)^N$ where N are the number of sensors. The Basic reconstruction algorithm is a particular case of the Viterbi List one, that saves one value for each sensor,($S = 1$, $1Best$). For the case of large networks, the amount of information to be processed is huge, so the VLA, even for 2Best is non-viable. Data increases exponentially with the number of sensors, so for example for N=20 and 2Bests 335.5MB of information have to be processed. The implementation of the algorithm in real cases is difficult with this large processing method, so we consider a variant of the Viterbi List algorithm that reduces the computational cost and achieves improvements over the simplest case 1Best (or BR).

5.1.3 Possible solution: Truncated Viterbi list algorithm

For the moment we consider only the 2Best case to implement the Truncated Viterbi List Algorithm (TVLA). The VLA accumulate large amount of data to analyze more combinations of it than the absolute ($1Best$) case (that only stores the lower-energy value for each sensor estimation), so we 're-start' the process in some point of the algorithm such that we can still have more combinations of data to analyze than in the absolute case. This implementation cuts the exponential cost increase by choosing only the two lower energy vectors of all the amount of data in one selected point. We denote the point of the algorithm wich re-starts the process as the *update* point. When the fusion center is estimating the best value for the $x[k]$, where $k = update$ the algorithm preserves only the two lowest energy samples calculated by that moment, after that, it launch the algorithm as this sensor was the first. In the tables 5.1 and 5.2 we can observe an example of the first iteration for both VLA and TVLA reconstruction in a 4 sensor network and 2Best. The final reconstruction for this iteration can be understood as $[a_1 a_2 a_3 a_4]$, where a_i is the

estimated value for the sensor i. For the TVLA $'*'$ and $'**'$ denotes the lower energy cases.

Table 5.1: Basic Viterbi List algorithm for 4 sensors (2Best)

S1	S2	S3	S4
5000	5100	5110	5113
4000	5200	5120	5112
	4300	5210	5121
	4400	5220	5122
		4330	5212
		4320	5214
		4490	5222
		4480	5226
			4332
			4331
			4323
			4322
			4492
			4491
			4483
			4482

Table 5.2: Truncated Viterbi List algorithm for 4 sensors (2Best) and $update = 2$

S1	S2	S3	S4
5000	5100	5210	5212
4000	5200*	4330	5214
	4300**		4332
	4400		4331

Its important to note that the information collected with few sensors is not enough to implement the Truncated Viterbi algorithm. In our scenario (20 sensors) we have relevant amount of information to avoid the full Viterbi algorithm and achieve benefits in terms of memory, speed

processing.

5.2 Minimization algorithms for high dimensional data

We propose two different algorithms for the high dimensional setup. For the Joint matching pursuit we relax the problem, since we assume that we perfectly know the transforms between views $F_{ij}(\cdot)$. For the joint matching pursuit we assume that we can approximate the transforms with certain error $\widetilde{F}_{ji} = F_{ji} + \epsilon$.

5.2.1 Joint total variation

We adapt the Chambolle algorithm for single image denoising for our joint method. To minimize the vector field G, Chambolle proposed to perform the minimization of the vector field in Eq.(4.11) with a fixed point iteration.

$$G_i^{l+1} = Proj\left(\widetilde{x_i^l} - \tau\nabla''_{x_i}\left(div(G_i^l) - \frac{y_i}{\lambda}\right)\right).$$

As a result of the variations in the minimization equation in Eq.(4.12) the gradient definition changes to $\nabla''_{x_i} = \frac{\alpha\nabla_{x_i}+\beta\nabla' x_i}{\alpha+\beta}$, using the cross gradient $\nabla' x_i$ defined in 4.13. $Proj$ is the orthogonal projector on the constraint $\|G_i\| \leq 1$. Usually, τ value is set to be $\tau < \frac{2}{\|\nabla \cdot div\|} = \frac{1}{4}$. The signal reconstruction is then $\widetilde{x_i} = y_i - \lambda div(G_i^L)$, where L is the number of final iterations. To fix L, we have to guarantee the energy convergence of the second term in the reconstruction ecuation $\lambda div(G_i^l) \to 0$ if $l \to L$.

The gradient is a directional change in the intensity in an image, so with the incorporation of the cross-images gradient $\nabla' x_i$ we can extract information from the sensor i neighborhood to increase the denoise efficiency. We add information to compute the horizontal and vertical variations of an image i, thus in a concrete pixel located in $(m,n)_i$, the variations with the classical gradient are function of $Var = f((m+1)_i, (n+1)_i)$. Otherwise if we introduce,in addition, the cross-image gradient $Var = f((m+1)_i, (n+1)_i, (m+1)_j, (n+1)_j)$ for all $j \in \mathbb{N}_i$. Note that the α and β parameters are a weighted average of the variance bring by the own signal and the neighbors respectively. Therefore, there is a tradeoff between them that have to be studied carefully.

Obviously in the definition of $\nabla' x_i$ there is implycity the function F_{ji} to find the equivalent pixel from image i to j, $F_{ji}[(m,n)_j] \to (m,n)_i$.

We can resume our JTV algorithm as:

Algorithm 1 Joint Matching Pursuit algorithm

$G_i = (G_x, G_y) = 0$ $i = [1; ...; nS]$
for $k = 1 \rightarrow L$ **do**
  for $i = 1 \rightarrow nS$ **do**
    $D_i = div(G_i)$
    $sG_i = \nabla''(D_i - \frac{y_i}{\lambda})$
    $G_i = G_i + \tau G_i$
    $G_i = \frac{G_i}{\|G_i\|}$
  end for
end for
$\widetilde{x_i} = y_i - \lambda div(g_i)$

5.2.2 Joint Matching Pursuit

We modify the classical matching pursuit algorithm to take advantage of the knowledge of the relative positions amongst sensors and the signal correlations within a neighborhood. First we introduce some notation: Z is the number of atoms that we use four our final reconstruction, L is the size of the dictionary and N is the number of sensors. F_i^m is the transformation of at the atom m for the image i. We define a set of N $test = [T_1, ...T_N]$, where each T_i contains the sensor i and the closest neighbors of i. With this, we want to exploit only the correlations between sensors that are within a certain spatial distance. Is logical to think that the differences increases as the distance does.

We use the transformation amongst views F_{ji} to enhance the performance of the algorithm. Obviously, we do not know exactly this transformations, although we can model it as $\widetilde{F}_{ji} = F_{ji} + \epsilon$, where ϵ denotes the uncertainty. MP is an iterative algorithm that computes the atom of the dictionary that best fits the residual image in each iteration. If m is the current iteration we approximate the transform as:

$$\widetilde{F}_{ji}^{*}m = F_i^{*m} - F_j^{*m}, \tag{5.1}$$

where F_i^{*m} denotes the transform of the atom that gives the highest projection coefficient until the iteration m. We define $A^m = \sum_{i=1}^{N} a_i^m$ as the best coefficient for the iteration m, where $a_i^(m)$ is the best projection for each view. Then, $A^{*m} = max[A^k]_{k=1}^{k=m-1}$ is the best coefficient until the iteration m, thus we use the associated transforms F_i^{*m} to compute the approximations $\widetilde{F}_{ji}^{*m}$ with Eq(5.2).

The first step of our algorithm is to compute jointly the coefficients and their associated

transforms for all the atoms within the different tests T_i, minimizing Eq.(4.14). We can organize the data in two matrices for both coefficients and transforms, where the rows are the atoms in the dictionary and the columns are the set of tests. We denote $Coeff$ and $Trans$ for the coefficients and transforms matrix respectively.

We can see the transform amongst images $\widetilde{F}^*_{ji}m$ as a 2 dimensional vector, mapping the distance 'x' and 'y' from one image to the other. The optimal transform lies within the range $\vec{F}_{ij} \in [(\widetilde{F}^*_{ij})_1 \mp \Delta F\ ,\ \widetilde{F}^*_{ij})_2 \mp \Delta F]$. As our transformation $\widetilde{F}^*_{ij}$ is not the ideal one, we have to take into account the possible variations within a certain range. We define f^{l_1,l_2}_{ij} where $l_1, l_2 \in [-\Delta F, +\Delta F]$ and $f^{l_1,l_2}_{ij} = [(\widetilde{F}^*_{ij})_1 \mp l_1\ ,\ [(\widetilde{F}^*_{ij})_2 \mp l_2]$. When we compute the correlation amongst the different neighbors in each test we have to take the maximum value for all the possible set of variations of a concrete transform $\widetilde{F}^*_{ij}$. Thus, given a concrete atom k of the dictionary on the iteration m, to estimate the proper transform approximation for each sensor we maximize the equation:

$$\widetilde{F^m_{ij}} = argmax_{l_1,l_2}\left\{\langle R^m y_i, g_{\gamma_k}\rangle + \lambda \sum_{j\in\mathbb{T}_i} f^{l_1,l_2}_{ji}(\langle R^m y_j, g_{\gamma_k}\rangle)\right\}. \tag{5.2}$$

Note that this maximization is the same as search jointly the maximum value of the projection within a certain window for all the neighbors.

Finding the maximum of the sum of the rows in $Coeff$ we select the best atom for the iteration m, this is A^m. If this coefficient is given by the atom (the row) k, we find the test which achieves the best projection within this row, $coeff(k,t) = maxcoeff(k,i)^{i=N}_{i=1}$. We fix the transformations of the sensors belonging to this test, $F_m\ \forall m \in T_t$. Then, we replace this transformations in all the test that have common neighbors with the test T_t. We repeat iteratively the process until the transforms of all the nodes are fixed. Before going through the next iteration, we compute the global approximation $F^*_{ji}m$ using Eq.(5.2).

We illustrate here an example of one entry of the *coor* matrix used in the algorithm:

$$coor(atom, test) = \begin{pmatrix} x & y \\ F^1_{ij_1}(x) & F^2_{ij_1}(y) \\ F^1_{ij_2}(x) & F^2_{ij_2}(y) \\ F^1_{ij_m}(x) & F^2_{ij_m}(y) \end{pmatrix}$$

Algorithm 2 Joint Matching Pursuit algorithm

while $m < N$ **do**
  for $n = 1 \to L$ **do**
    for $i = 1 \to N$ **do**
      $[coeff(n,i), coor(n,i)] = max_{(g_{\gamma_n}, l_1, l_2)} \left[corr\left(F_i^n(g_{\gamma_n}), R^m y_i\right) + \sum_{j \in \mathbb{T}_i} corr(F_j^n(g_{\gamma_n}), f_{ij}^{l_1,l_2}(R^m y_j)) \right]$
    end for
  end for
  for $n = 1 \to L$ **do**
    $MaxCof(n) = \sum_{i=1}^{nS} coeff(n,i)$
  end for
  $A^m = max(MaxCof)$
  if A^m is given by index k **then**
    $Atom = g_{\gamma_k}$
  end if
  $T_t = max(coeff(k, t \in [1, ..., N]))$
  for $l \in T_t$ **do**
    $F_l^m = coor(k, T_t)_l$
    $FixedCoor \leftarrow l$
  end for
  {Recalculate the coefficients of the different tests fixing the transforms F_l^m}
  while $FixedCoor < N$ **do**
    for $i = 1 \to N$ **do**
      if $i \notin FixedCoor$ **then**
        $[coeff(k,i), coor(k,i)] = max_{l_1,l_2)} \left[corr\left(g_{\gamma_k}, R^m y_i\right) + \sum_{j \in \mathbb{T}_i} corr(g_{\gamma_k}, f_{ij}^{l_1,l_2}(R^m y_j)) \right]$
      end if
    end for
    $[T_t] = max(coeff(k, t \notin FixedCoor))$
    for $l \in T_t$ **do**
      $F_l^m = coor(k, T_t)_l$
      $FixedCoor \leftarrow l$
    end for
  end while
  for $i = 1 \to nS$ **do**
    $Rec_i^m = \left\langle F_i^m(g_{\gamma_k}), R^m y_i \right\rangle F_i^m(g_{\gamma_k})$
    $R^{m+1} y_i = R^m y_i - F_i^m(g_{\gamma_k})$
  end for
end while
for $n = 1 \to L$ **do**
  $MaxCof(n) = \sum_{i=1}^{nS} coeff(n,i)$
end for
$A^m = max(MaxCof)$
if $A^m > A^{m-1}$ **then**
  for $i = 1 \to N$ **do**
    for $j = 1 \to N$ **do**
      $F_{ji}^* m = F_i^m - F_j^m$
    end for
  end for
end if

Chapter 6

Experimental setup

6.1 Algorithm and network parameters analysis for scalar sensor data

Our final goal is to simulate a 20 sensor network using the Truncated Viterbi algorithm presented before. Due to the computational cost, we work previously with less sensors (6 and 11) to see the general behavior of certain parameters of the minimization function. This allows us to formulate some hypothesis to work on in the 20 sensors case. As we explain in previous chapters, we implement the minimization algorithm with three different regularization techniques: the absolute value (ABS), total variation (TV), and weighted distance (WD). In low sensors scale we use the classic VLA to see the general behavior amongst different parameters, then we can use this information for the TVLA for 20 sensor network. Concretely, we notice that the maximum PSNR is achieved for similar λ values for 1Best, 2Best and 3Best for 6 and 11 sensors, so we estimate this parameter for the 20 sensor network and TVLA simulating only the 1Best case. This is useful because the parameter analysis for 20 sensor networks and NBest for $N > 1$ is computationally non-viable.

In practice, most of the regularizations of the inverse problems suffers from a trade-off between the data-fidelity-term and the smoothness term. Different regularization techniques deal with this trade off. This trade off can be controlled by the selection of proper regularization parameter. Various methods have been developed for the optimal selection of these regularization parameters [14], but the goal of this work is not the optimum selection of λ. Our discrete minimization algorithm differs a bit on the classical ones, so is not easy to compute this type of numerical analysis. We select λ by simulating the data reconstruction performance in a large range of this

parameter. To see the process convergence, we average the results over 150 different experiments. We change randomly the sender nodes and the receiver sensors in the graph model. The set of experiments for 6, 11 and 20 sensors we illustrate in appendix A. We conclude that for $\lambda = 0.75$ and TV regularization, TVLA is achieving the best PSNR results.

As we mention in section 5.1.3, another key parameter is update point (*update*), because this is the parameter which controls the trade off between the data processed and the computational cost. As the information computed increases the PSNR does the same, but it's important to find a point where the time spent in the process is reasonable.

As we can see in A.1, the computational cost is progressively decreasing from the eight sensor estimation, but then increases again from the twelfth. This shows that if we cut the algorithm when the fusion center is processing the first or last sensors approximations the amount of data is so large to be processed efficiently. We have to find a intermediate point where the exponentially increasing of information does not make our algorithm too slow. The PSNR values will be showed deeply on 7, but the best updates points are $Update = [8, 10]$. So obviously, the choice will be $update = 10$ because the time spent in the process is much lower.

Finally, the topology graph and the data model for 20 sensors is represented in figure A.9. For this, we use a 2D-Gaussian with variance , with the values varying between [7...15].

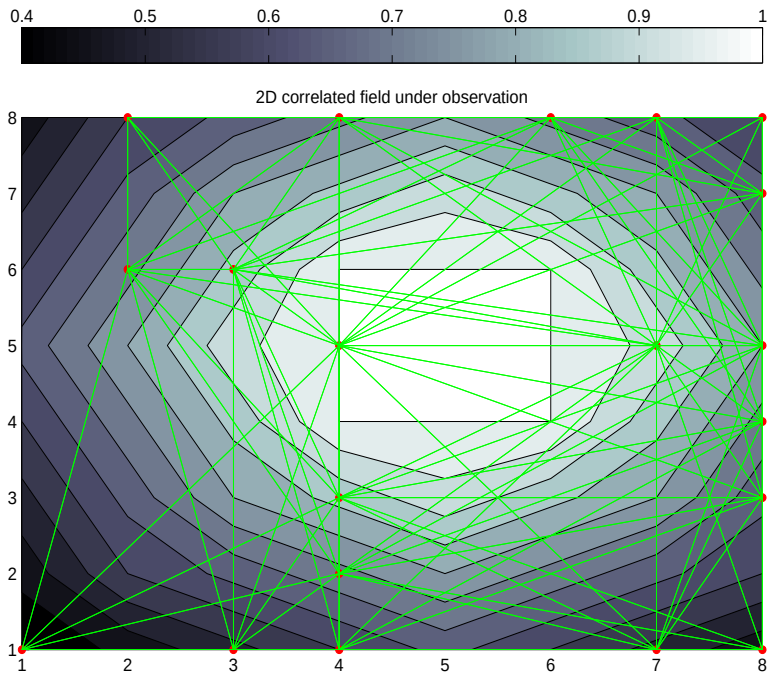


Figure 6.1: Graph and data model for 20 sensors

6.2 Algorithm and network parameters analysis for high dimensional sensor data

We work with a twelve sensor network, we can see the topology graph in 6.2.

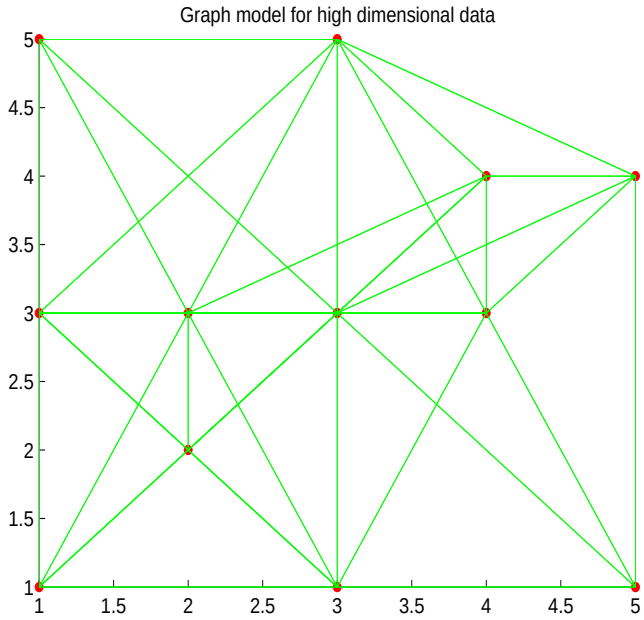


Figure 6.2: Graph model for 20 sensors

6.2.1 Joint Matching Pursuit

As we explained before, our input signal is a sparse image composed by three atoms. We built artificially the dictionary, using a 2D-view generator software. We denote each element of this dictionary as g_{γ_k} for $k = [1, ..., 33]$. We incorporate in the dictionary eleven scaled versions of each atom that generates the input data. Note that each element of the dictionary is an image of the same size as the input data which contains an atom in the center of it. So g_{γ_k} is not the full dictionary, we use the full dictionary in the MP algorithm translating the atoms in each position of the view. Therefore if our image size is 64×64 pixels, we use $64 \times 64 \times 33$ atoms to reconstruct the signal. So the transforms for an atom m for the sensor i, $F_i^m g_{\gamma_k}$ can also be understood as the displacement of a certain atom to fit the original image. We can see an example of the atoms

included in our dictionary in picture.

In this work we compare two methods of MP algorithms: the single matching pursuit (SMP) and our method, the joint matching pursuit (JMP). The first one, decodes independently the signal of each sensor without using any information about the neighbors or its relative position in the network. The second one, decides the atom that match best the signal in each iteration by projecting it along the image residues of all the sensors. Our algorithm JMP as we explained in the previous chapters exploits the correlations amongst views as well as the knowledge of the relative positions of nodes in the network.

6.2.2 Joint Total Variation

On the same context, we compare our method with the classical total variation one, which we denote as single total variation (STV). This technique minimizes the total variation of each image independently, without assuming any prior of the network topology or the correlation within neighbor views.

For our joint total variation algorithm is necessary to fix the regularization parameter λ and the constants α and β, which control the contributions of neighbors information and the own sensor image. We use an experimental method to approximate a proper value for this parameters. We simulate the algorithm varying the constants within a certain range, as we did in the scalar case for the regularization parameter λ. Due to the lack of relevance we omit this plots, and we give only the final results. The best performance is achieved for λ=0.125, α=1 and β=5. Obviously, we use the same λ to contrast both STV and JTV methods.

To exploit the correlation between neighbors, we work with different measurement matrix for each sensor image. Therefore the matrix Φ in equation Eq.(3.6) varies in function of the node in the network, Φ_i, $i = [1, ..., N]$, where N is the number of sensors. The fact of having different measurement matrices, implies that the information from the neighbors is more relevant. If we compress the signal with a constant Φ for all the sensors the observations within a neighborhood becomes more redundant.

Chapter 7

Final Discussion

7.1 Scalar data discussion

We gather information of both 2 and 3 Best only for the 6 sensor network case, due to the computation limitations. It's interesting to see that in the 6 sensor network for 2 and 3 best the method achieving the best PSNR is the WD, despite being deficient for 1Best (see table 7.1). We allow less variance on data as the number of sensors increases, and for more nodes the smoothness of data is better characterized. As a result, TV estimates better the signal under this assumptions (20 sensors) as we can see in table 7.3. Otherwise, for 6 sensors where data can change a bit more within a neighborhood TV results are the worst.

We discard the regularization methods ABD and WD for 20 sensors due to the poor results obtained for the 1Best simulations. The 2Best implementation with TVLA, is an extension of the simple VLA for 1Best, so is logical to think that if the basic construction is not achieving acceptable PSNR, the extension will do not too.

Table 7.1: PSNR (dB) for 6 sensors with the classic Viterbi List algorithm

	1Best	2Best	3Best
WD	18.1	23.2	25.3
ABS	19.5	22.8	22.9
TV	19.4	21	21.3

Table 7.2: PSNR (dB) for 11 sensors with the classic Viterbi List algorithm

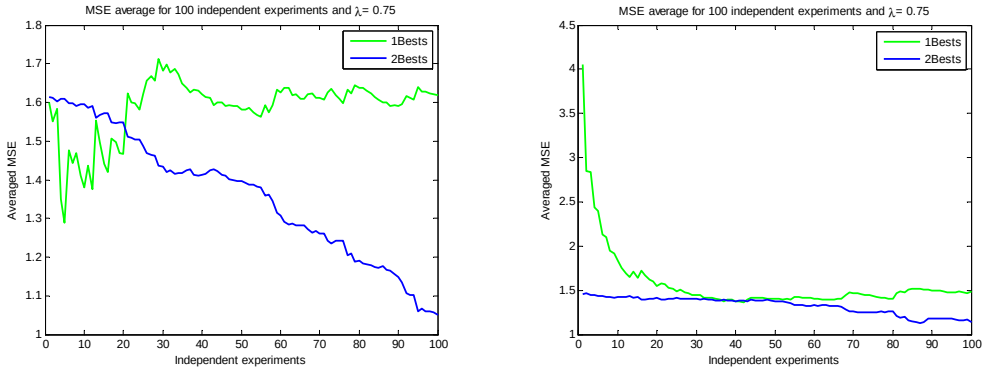
	1Best	2Best
WD	20.9	22.2
ABS	23.4	24.7
TV	19.3	19.7

Table 7.3: PSNR (dB) for 20 sensors with the classic Viterbi List algorithm

	1Best
WD	13.9
ABS	18.3
TV	22.2

7.1.1 PSNR and convergence for TVLA

After the simulations of low sensor networks and analyze the results to know the behavior of the final 20 sensor network, we can obtain relevant information for the selection of our regularization parameter $\lambda = 0.75$ for TV. It's important because the quality of the final reconstruction is extremely related with this value. We also decide the *update* parameter knowing that the computational cost of the Viterbi List algorithm is exponentially increasing with the number of sensors. As we can see in table 7.4 TVLA for 2Best achieves better PSNR compared to BR defined in 5.1.1. Remember that BR is equivalent to the VLA for 1Best. If we select *update*=8 the improvement in the PSNR is around 1.1 dB, on the other hand the increase is 0.7 dB for *update*=10. The improvement in dB for $update = 8$ compared to $undate = 10$ is 0.4 but the time spent in the entire reconstruction process is more than two times larger. So if our framework do not have time or computation limitations and the main goal is to achieve the best PSNR as possible, the best update choice will be $update = 8$. Otherwise, as the increase in the PSNR is not extremely relevant, the best option for applications where the computational cost has to be considered is $update = 10$. The other point showed in 7.1(a) and 7.1(b) is that for $update = 8$, the algorithm convergence is slow, so for few experiments the results are more random.



(a) (b)

Figure 7.1: (a) Convergence of 1Best VLA and 2Best TVLA for $update = 8$, (b) Convergence of 1Best VLA and 2Best TVLA for $update = 10$

7.1.2 Sumary for TVLA vs VLA

We can consider that we can represent a wide ride of temperatures with 6 bits. The implementation of the VLA for 20 sensors and 2Best requires in terms of memory:

$$\sum_{i=1}^{N} s^i \times N \times b, \tag{7.1}$$

where N is the number of sensors, s the elements saved in TVLA and b the bits used. For our study 31.4MB. On the other hand if we use the TVLA the computational cost can be calculated as

$$\left(\sum_{i=1}^{u-1} s^i + \sum_{i=1}^{N-(u-1)} s^i \right) \times N \times b, \tag{7.2}$$

where u is the $update$ point. For this case the amount of data processed is 161KB for $update = 8$ and 40.6KB for $update = 10$. This means that we use the 0.55% and 0.13% of the information that is required for VLA, in terms of computational cost is a great improvement.

7.1.3 Conclusions of TVLA for 2Best

Is clear that 2Best TVLA is better that 1Best VLA (or BR). If we analyze the energy cost function Eq.(4.1) is easy to conclude that with our algorithm we consider more points of the function domain to compute the energy minimum. These increasing of computational burden

Table 7.4: TVLA for 20 sensors (2Best)

	$Update8$	$Update10$
PSNR(dB)	23.3	22.9
Improvement respect BR (dB)	1.1	0.7
Time Spent (sec)	1000	300
Data needed (KB)	161	40.6
Data used respect VLA (%)	0.55	0.13

or what is the same, the higher number of possible sensor values processed, is beneficial for the localization of the absolute energy minimum.

7.2 High dimensional data discussion

7.2.1 Joint Total Variation

We try to compare the PSNR and the final reconstruction images for different number of measurements with both classical single total Variation (STV) and our method, the Joint Total Variation (JTV). We did the experiments with a range of measurements M, varying from the 20% to the 60 % of the total image size P.

If we observe the PSNR plot in 7.2, the JTV model is achieving best results for the full range of measurements. But the PSNR in this case is not the best representation of the algorithm performance. If we observe the final reconstruction images, we realize that the improvement in terms of PSNR is not reflected in the final quality of the image. We can not distinguish visually the differences between both methods, so we can conclude that our algorithm is not exploiting as much as we expect the information of the neighborhood nodes.

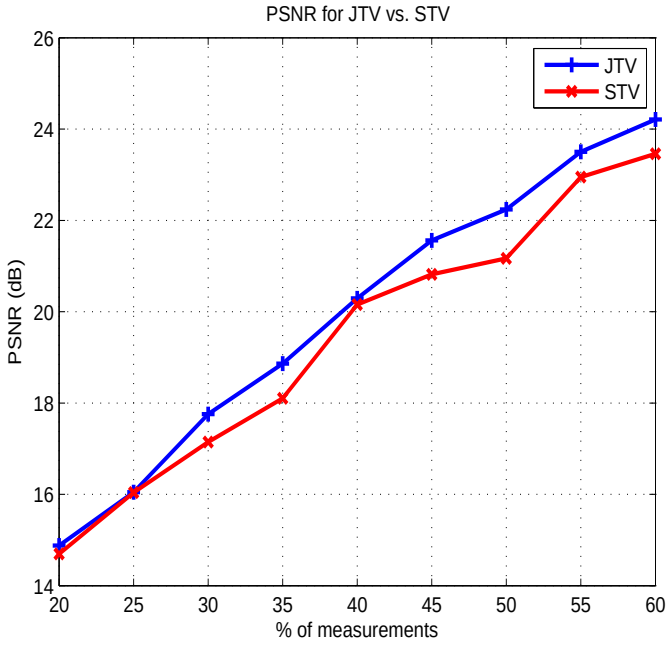
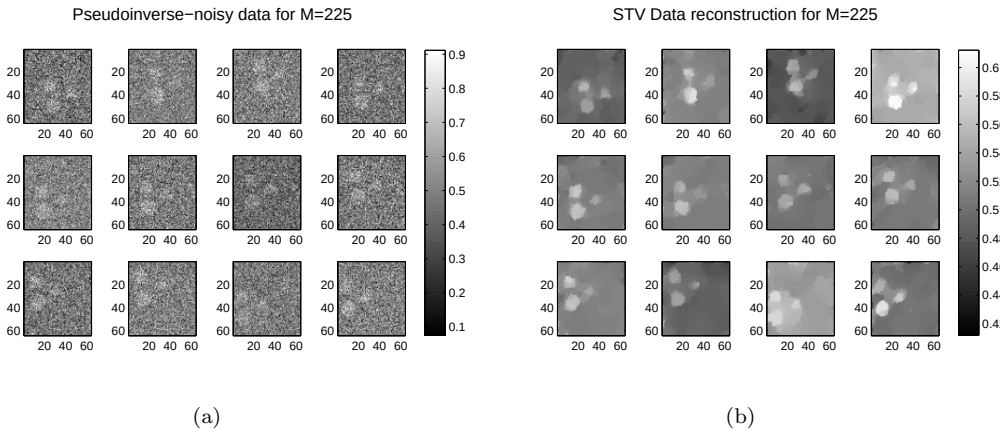
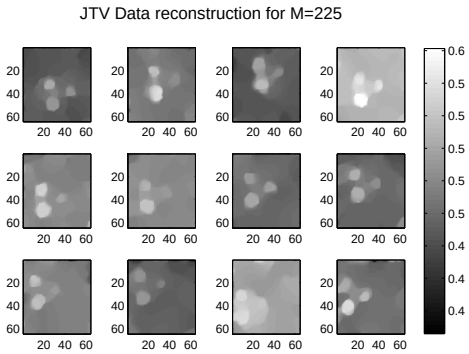


Figure 7.2: PSNR for STV and JTV

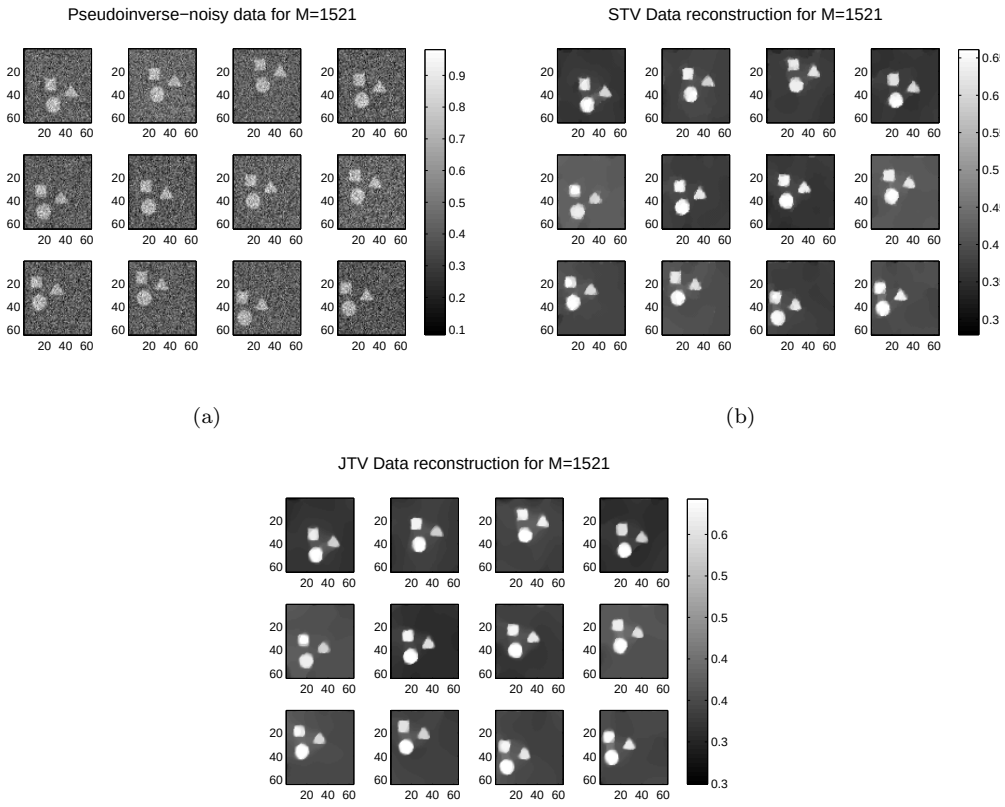


(a) (b)



(c)

Figure 7.3: Reconstruction for both STV and JTV methods. (a) Noisy images for M= 15 × 15 measurements , (b) STV for M=15 × 15 measurements, (c) JTV for M=15 × 15 measurements



(a) (b)

(c)

Figure 7.4: Reconstruction for both STV and JTV methods. (a) Noisy images for M= 39 × 39 measurements , (b) STV for M=39 × 39 measurements, (c) JTV for M=39 × 39 measurements

7.2.2 Conclusions of JTV

Our noisy images for this case are the result of 'decompress' the signal. The total variation of an image have to be computed in the signal domain, so when we multiply the observations by the pseudo inverse of the measurement matrix $\widetilde{x}_i = \Phi^t y_i$ the 'noise' added don't allow the method work properly. When we work with a low compressing ratio as we can see in figure 7.4 the method reconstructs the signal with an acceptable quality, but there is not any significant difference between STV and JTV. By the other hand, when we reduce the number of measurements both methods obtain a poor representation of the original image, see figure 7.3.

7.2.3 Joint Matching pursuit

We want to compare our joint matching pursuit (JMP) algorithm with the classical single matching pursuit (SMP), which decodes independently each image. We test the algorithms for set of measurements M varying from the 15% to the 55% of the total number of pixels in the image.

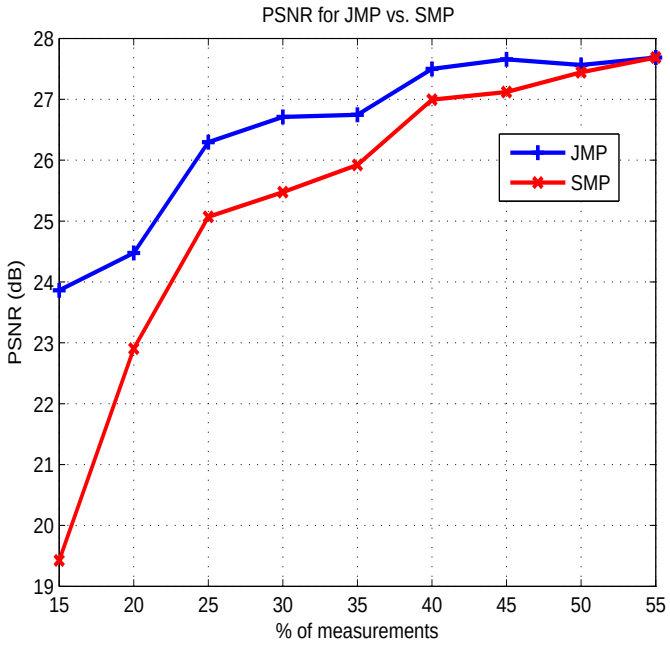


Figure 7.5: PSNR for SMP and JMP

As we can see in fig. 7.5 our proposed algorithm achieves significant improvements in terms of PSNR for the low range of measurements. From M=35% to 55% of the mesureaments the methods converge to the same value, since the number of measurements is already high enough.

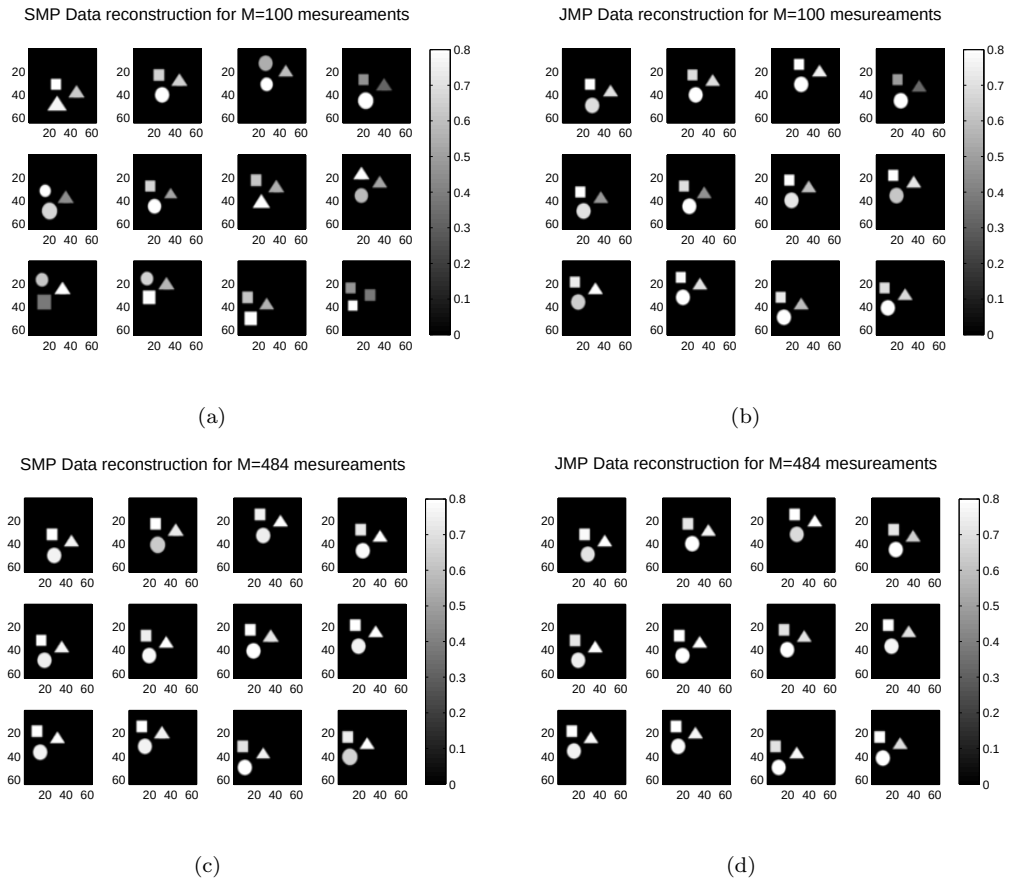


Figure 7.6: Reconstruction for both SMP and JMP methods. (a) SMP for M=10 $\times$ 10 measurements , (b) JMP for M=10 $\times$ 10 measurements , (c) SMP for M=20 $\times$ 20 measurements , (d) JMP for M=20 $\times$ 20 measurements

In figure 7.6(a) SMP includes a lot of wrong atoms in the decoded image. The transformations of this atoms are clearly far away from the ideal ones. Otherwise in figure 7.6(b), there are also some mistaken atoms and transformations but the error is clearly lower compared to the SMP. Note that for JMP we recover three different atoms for all the signals and their relative position. Instead JMP enhance the final PSNR, the final reconstruction do not produce relevant visual differences when we increase the number of measurements, see figure 7.6(c) and figure 7.6(d).

Despite the improvements achieved, our method fails when M <15% of the image pixels. The JMP algorithm approximates the transformations amongs views $\widetilde{F}_{ij}(\cdot)$ in each iteration after the computation of the best atom for the full set of images. The estimation of this transforms constraints a lot the performance of the algorithm. We want to see the behavior of JMP if we

assume that we perfectly know the transformations $F_{ij}(\cdot)$. This method can be viewed as an upper bound of the JMP and we denote it Ideal Joint Matching Pursuit (IJMP).

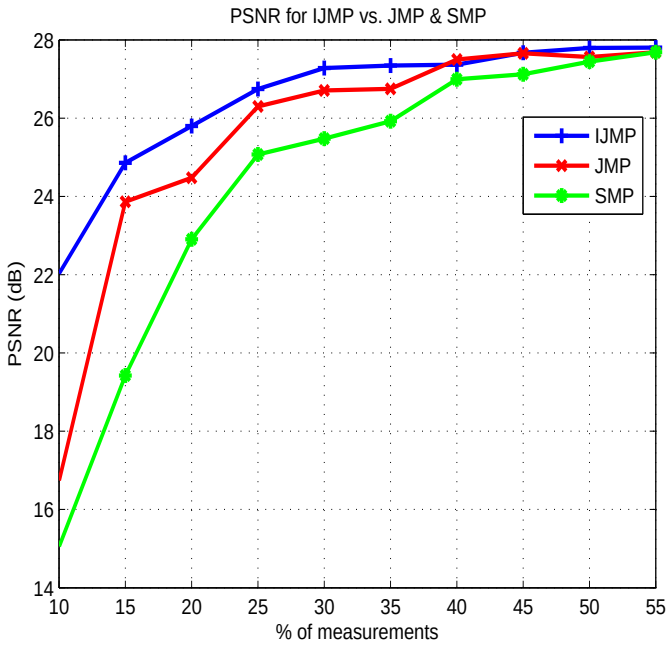
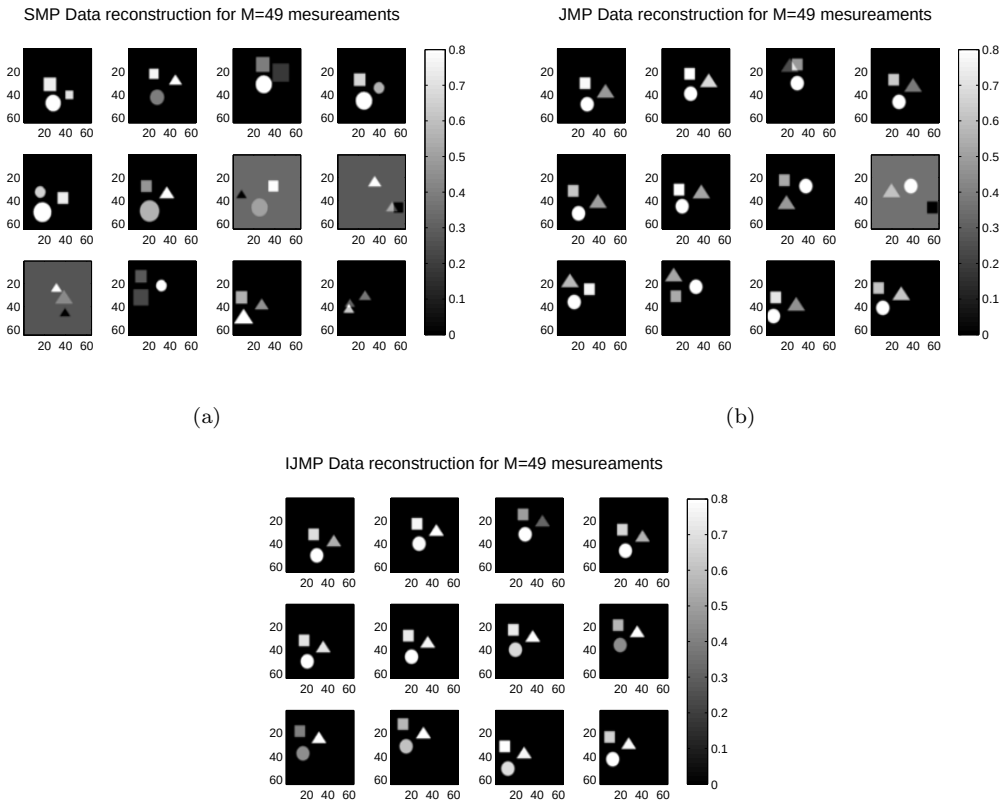


Figure 7.7: PSNR for STV,JTV and IJMP

As we can see in figure 7.7 IJMP enhances a lot the PSNR for low values of M, about $5dB$ for 49 measurements over 64×64 image pixels. On the other hand, the reconstructed signals illustrate clearly the benefits of the algorithm if we increase the quality of the transformations amongst views.



(a) (b)

(c)

Figure 7.8: Reconstruction for SMP, JMP and IJMP methods. (a) SMP for M=7 × 7 measurements , (b) JMP for M=7 × 7 measurements , (c) IJMP for M=7 × 7 measurements.

If we observe the decoded signals in figures 7.8(a), 7.8(b) and 7.8(c) we corroborate visually the results of the PSNR plot. IJMP estimates the correct atoms, but the transforms F_{ii} of this atoms are not perfectly recovered. Otherwise SMP and JMP do not recover correctly either the atoms or the transforms.

7.2.4 Conclusions of JMP

Our method is able to decode compressed signals for lower number of measurements than the SMP. Thus, it exploits the correlation amongst a neighborhood to increase the performance of the algorithm. Despite the improvement compared to SMP, our solution for M¡15% of the image size, do not achieve acceptable results. We use IJMP to prove empirically that with better estimations of the transformations amongst views, the reconstruction quality enhances for lower M. We can

conclude that the technique used to approximate this transforms is the main handicap of our algorithm. If we were able to estimate $\widetilde{F}_{ij}(\cdot)$ with less error the quality of JMP would increase.

Bibliography

[1] R. Baraniuk, V. Cevher, M. Duarte, and C. Hegde.
"Compressive Sensing", 2007.

[2] D. Kempe, A. Dobra, and J. Gehrke.
"Gossip-Based Computation of Aggregate Information", 2003.

[3] S. Boyd, A. Ghosh, B. Prabhakar and D. Shah.
"Randomized Gossip Algorithms", 2006.

[4] C. Aysal, E. Yildiz and A. Scaglione.
"Broadcast Gossip algorithms", 2008.

[5] P. Kyasanur, R. Choudhury and I. Gupta.
"Smart gossip: An adaptive gossip-based broadcasting service for sensor networks", 2006.

[6] G. Dimakis, S. Kar, M. Moura, F. Michael, G. Rabbat, and A. Scaglione.
"Gossip Algorithms for Distributed Signal Processing", 2010.

[7] C. Luo, F. Wu, J. Sun and C. Chen.
"Efficient Measurement Generation and Pervasive Sparsity for Compressive Data Gathering", 2010.

[8] C. Luo, F. Wu, J. Sun and C. Chen.
"Compressive Data Gathering for Large-Scale Wireless Sensor Networks", 2009.

[9] J. Wright, M. Yi, J. Mairal, G. Sapiro, T.S. Huang and S. Yan.
"Sparse Representation for Computer Vision and Pattern Recognition", 2010.

[10] A.Y. Yang, M. Gastpar, R.Bajcsy and S.S. Sastry.
"Distributed Sensor Perception via Sparse Representation", 2010.

[11] W. Wang and K. Ramchandran.
"Random distributed multiresolution representations with significance querying", 2006.

[12] G. Dimakis, D. Sarwate and J. Wainwright.
"Geographic gossip: efficient aggregation for sensor networks", 2006.

[13] W. Wang, M. Garofalakis and K. Ramchandran.
"Distributed sparse random projections for refinable approximation", 2007.

[14] V. Agarwal.
"Total Variation Regularization and L-curve method for the selection of regularization parameter", 2003.

[15] D. Zhou and B. Scholkopf.
"Regularization on Discrete Spaces", 2005.

[16] E. Candes and T. Tao.
"Decoding by linear programming", 2005.

[17] D. Donoho.
"Compressed sensing", 2006.

[18] E. Candes, J. Romberg and T. Tao.
"Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information", 2006.

[19] M. Bertero, C. De Mol and E. R. Pike.
"Linear inverse problems with discrete data. I. General formulation and singular system analysis", 1985.

[20] M. Bertero, C. De Mol and E. R. Pike.
"Linear inverse problems with discrete data: II. Stability and regularisation", 1988.

[21] S. S. Chen, D. L. Donoho, and M. A. Saunders.
"Atomic Decomposition by Basis Pursuit", 1998.

[22] S. G. Mallat and Z. Zhang.
"Matching pursuits with time-frequency dictionaries", 1993.

[23] P. Frossard, P. Vandergheynst, R.M. Figueras i Ventura, and M. Kunt.
"A posteriori quantization of progressive matching pursuit streams", 2004.

[24] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani.
"Least angle regression", 2004.

[25] R. Tibshirani.
"Regression Shrinkage and Selection via the Lasso", 1996.

[26] A. Chambolle and P. Lions.
"Image recovery via total minimization and related", 1997.

[27] A. Chambolle.
"An Algorithm for Total Variation Minimization and Applications", 2004.

[28] J. Besag.
"Spatial Interaction and the Statistical Analysis of Lattice Systems", 1974.

[29] Blender software.
"http://www.blender.org".

Appendix A

Appendix title

Our final goal is to simulate a 20 sensor network using the Truncated Viterbi algorithm presented before, but due to the computational cost, we work previously with less sensors (6 and 11) to see the general behavior of certain parameters of the minimization function. This will allow us to formulate some hypothesis to work on in the 20 sensors case.

A.1 6 Sensors network

We can see the topology graph and the data model in A.1. We design a random geometric graph, of the size 6×6, where all the sensors are initially placed. An edge between two nodes is added if they are positioned within a communication radius r of each other. As proposed in [11] the critical threshold above which the graph is connected is $\Theta\left(\sqrt{\frac{\log n}{n}}\right)$. As the computational cost of this case is not too high, we allow a large variance on data, when building matrix field. For this, we use a 2 dimensional Gaussian with variance $\sigma \in [4, ..., 29]$.

As we explain in previous chapters, we implement the minimization algorithm with three different regularization techniques: the absolute value (abs), total variation (TV), and weighted distance (WD). In low sensors scale we use the classic VLA to see the general behavior amongst different parameters, then with this information we can improve the performance of the TVLA for 20 sensor network. Concretely, if we observe that the maximum PSNR is achieved for similar λ for 1Best, 2Best and 3Best for 6 and 11 sensors, we can estimate this parameter for the 20 sensor network and TVLA simulating only the 1Best case. This will be useful because the parameter analysis for 20 sensor networks and NBest for $N > 1$ is computationally non-viable.

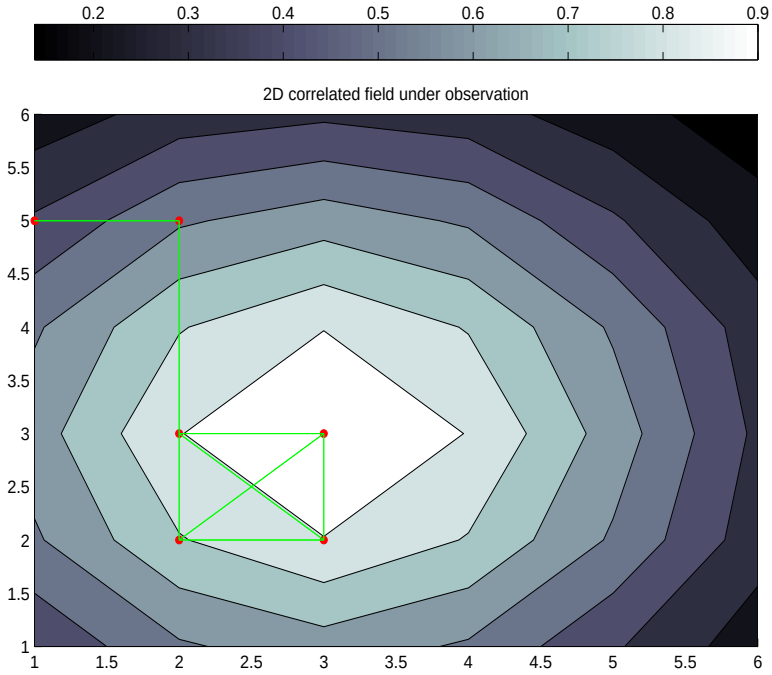


Figure A.1: Graph and data model for 6 sensors

A.1.1 Lambda selection

In practice, most of the regularizations of the inverse problems suffers from a trade-off between the size of the regularized solution and the quantity of the fit that it provides to the given data. Different regularization techniques differ on the basis on how they minimize this trade off. This trade-off can be controlled by the selection of proper regularization parameter. Various methods have been developed for the optimal selection of these regularization parameters [14], but the goal of this work is not the optimum selection of λ, due to our discrete minimization algorithm differs a bit on the classical ones. Thus, is difficult to compute the numerical analysis proposed in [14] for the regularization parameter selection . We select the proper λ simulating the network gathering process in a large range of this parameter. To see the process convergence, we average the results over 150 different experiments, this is, we change randomly the senders and the receivers in the graph model.

As we can see in A.4 WD method is achieving best results for 6 sensors, maybe because we allow more variance on data as the computational cost is low for 6 sensor case. The other key point is that in A.3 the PSNR peak for 1Best and TV is shifted, so is important to consider it for the λ selection in higher sensor networks. Note that the improvement on the SNR between 1Best

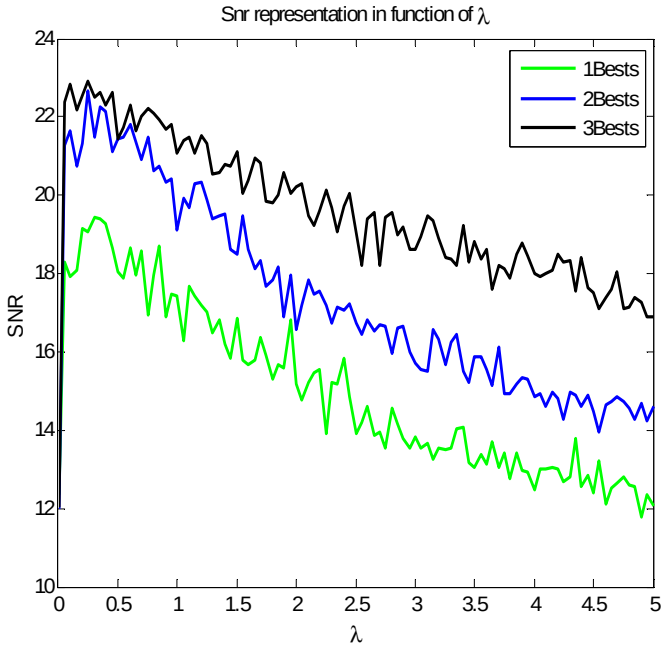


Figure A.2: Abs regularization

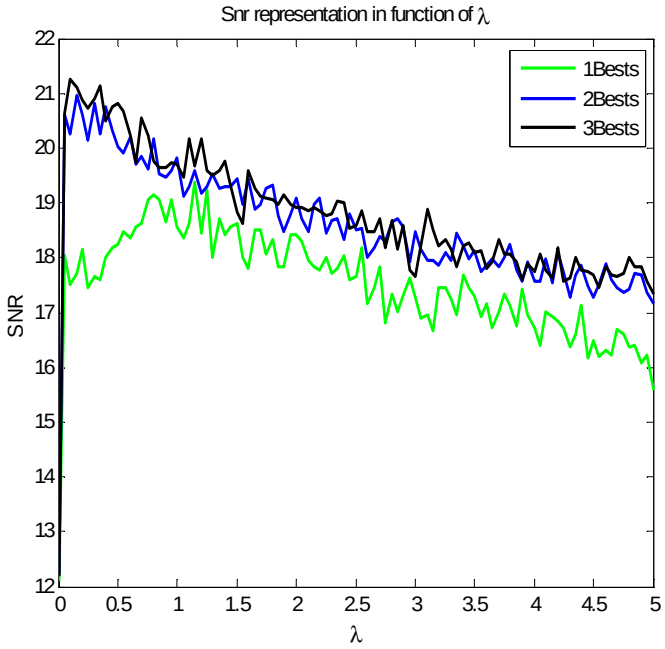


Figure A.3: TV regularization

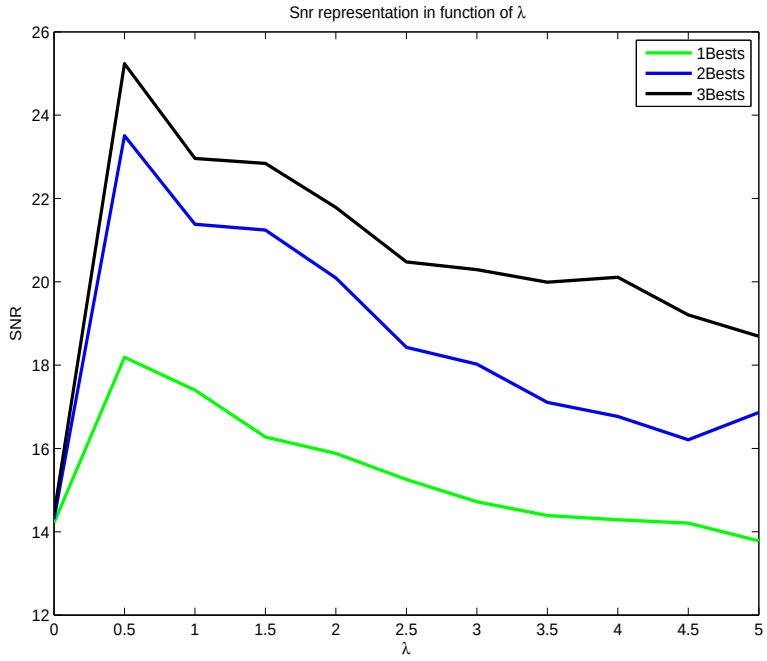


Figure A.4: Wd regularization

and 2Best is much higher than 2-3Best. If the behavior for more sensors were the same, will be reasonable to consider only the 2Best case because the tradeoff between PSNR and computational cost is not acceptable for 3Best.

A.2 11 Sensors network

The graph was constructed with the same criterion explained before for 6 sensor network. Now the data is modeled as a 2 dimensional Gaussian with variance $\sigma \in [7, ..., 24]$,we can see both data and graph model in A.5.

A.2.1 Lambda selection

In this case, the 3Best case is already too computationally expensive, this is the reason why we realize the experiments for 1 and 2Best, to observe the behavior and extrapolate conclusions for the final 20 sensors scenario.

In A.8 the SNR peaks between 1 and 2Best are also shifted, so this fact must be considered for higher sensors network.

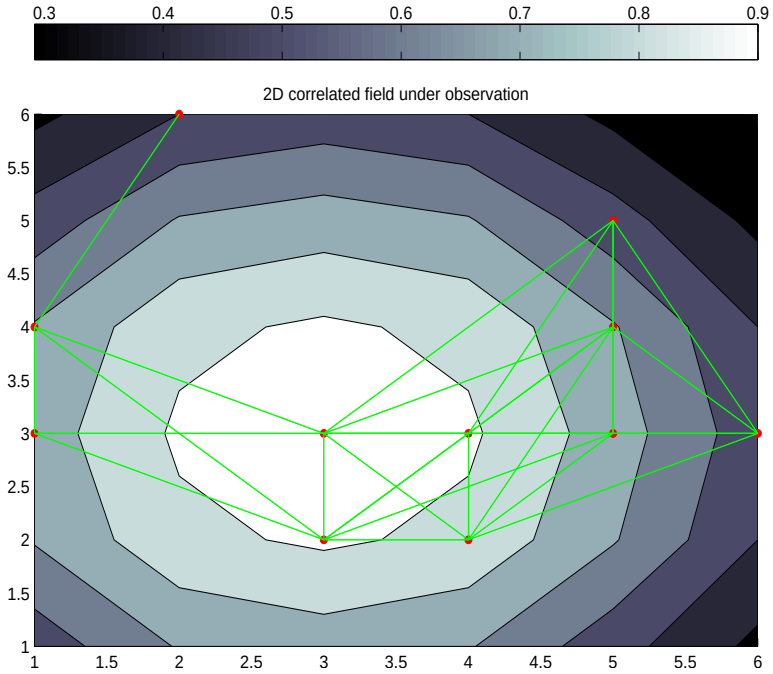


Figure A.5: Graph and data model for 11 sensors

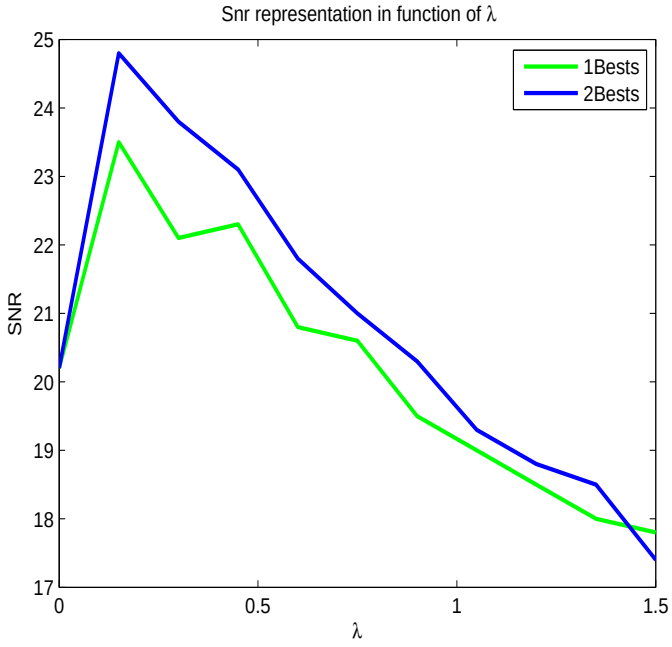


Figure A.6: Abs regularization

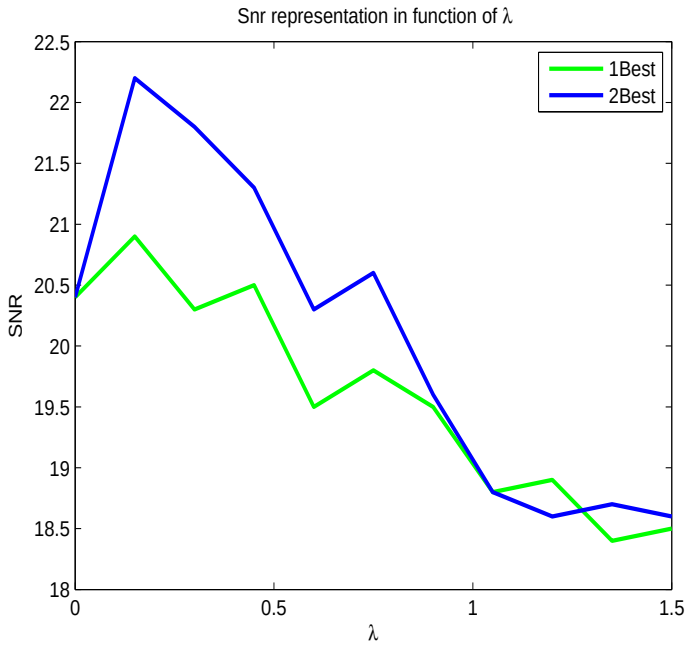


Figure A.7: Wd regularization

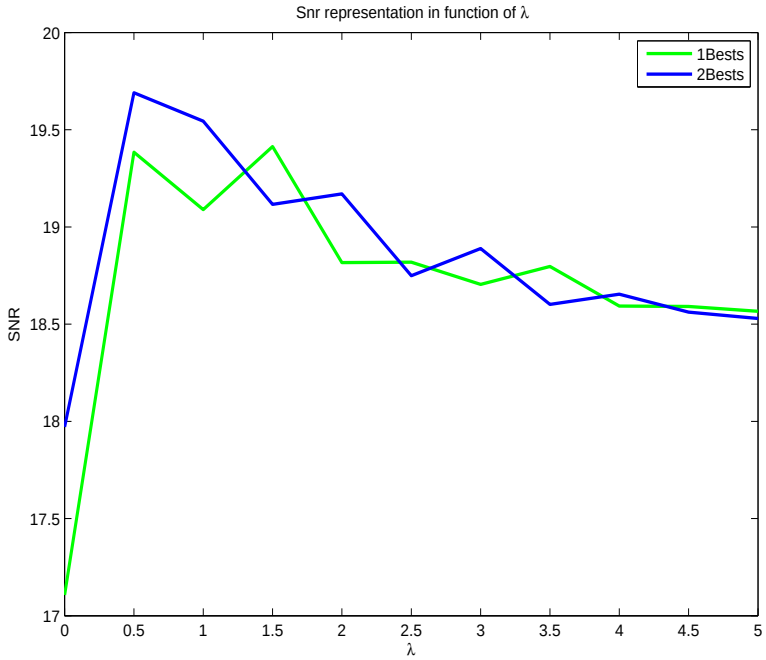


Figure A.8: TV regularization

A.3 20 Sensors network

Now the data is modeled as a 2 dimensional Gaussian with variance $\sigma \in [7, ..., 24]$. We can see both data and graph model in A.9

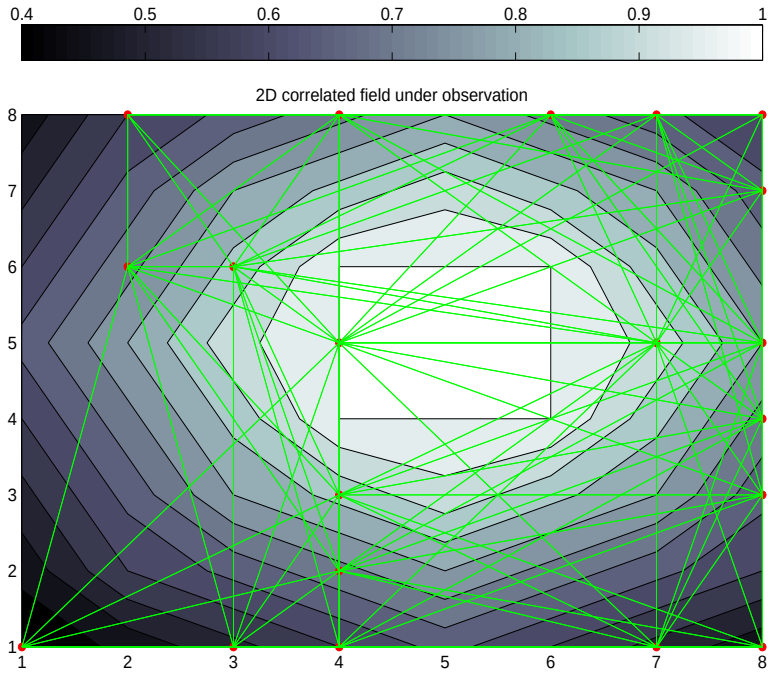


Figure A.9: Graph and data model for 20 sensors

A.3.1 Lambda selection

First of all we have to select the optimal lambda. For computational cost reasons, we experiment only with 1Best VLA and later using the conclusions deducted from 6 and 11 sensors networks we can approximate λ for 2Best and TVLA.

For 20 sensors, there have more samples of data, so the smoothness is represented with more fidelity. We can observe that TV is clearly achieving the best PSNR values, abs regularization have good behavior with less sensors but if we increase the number of nodes this method can not be considered as a real alternative. We avoid the WD regularization because is the worst method and we want to focus our efforts on the study of TV. Is reasonable to think that the best regularization for 1Best case will be also the most appropriate for the implementation of our 2Best TVLA algorithm. Note that for this simulations of 1Best we can still use the VLA, since we want

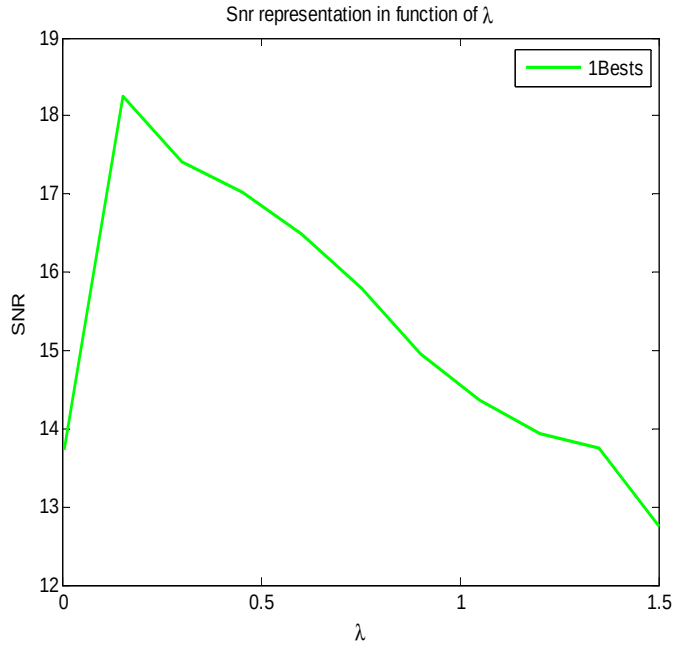


Figure A.10: Abs regularization

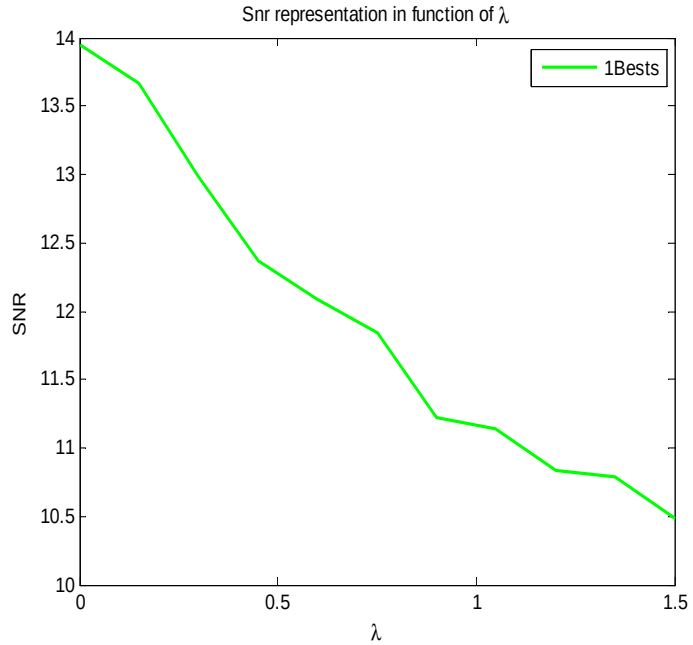


Figure A.11: Wd regularization

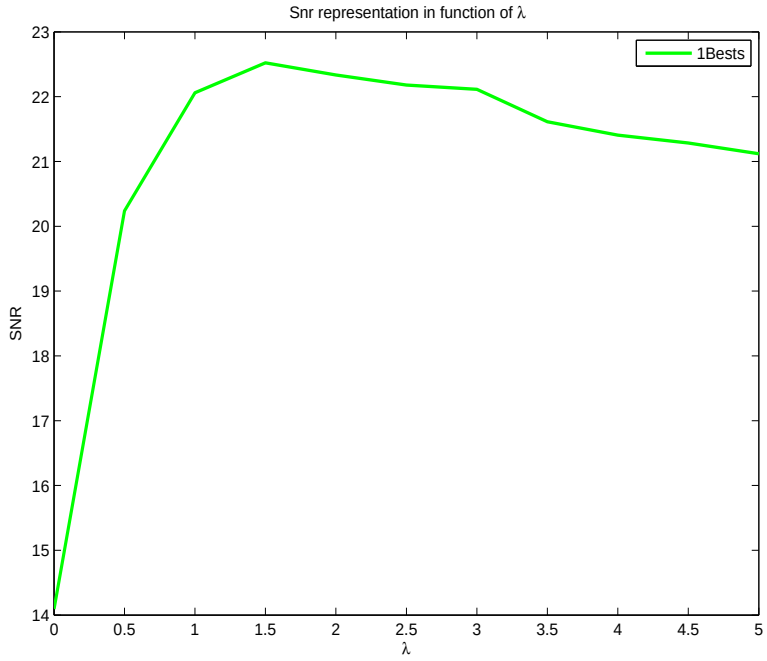


Figure A.12: TV regularization

to represent the optimal lambda for 1Best and avoid the large amount of information required in 2 or 3Best. For TV the PSNR peak is shifted from 1Best to 2Best for 6 and 11 sensors. The selection of λ depends on the norm of the reconstructed signal (exactly on the l_2 norm). We can use Eq.(A.1) to calculate λ for TV and 2Best in the 20 sensor network.

$$\lambda \approx \frac{d_{12}\,|x|_{6-11s}}{|x|_{20s}} \tag{A.1}$$

In A.1 d_{12} is the averaged distance between 1-2Best peaks for 6 and 11 sensors, $|x|_{6-11s}$ is the averaged norm of reconstructed signal for 6 and 11 sensors. Analogously, $|x|_{20s}$ is the norm of the reconstructed signal for 20 nodes. Using A.1 we can find a acceptable value for the constant under analysis, $\lambda = 0.75$.

A.3.2 Update point for the Viterbi List algorithm

Since the increase of required computational resources of data is exponential, if we cut the algorithm while approximating the first sensors values or the lasts, the time processing will be still unacceptable. We perform experiments by re-starting the process between the eighth and four-

Table A.1: Viterbi algorithm's time analisis along different *update* values

Update	8	9	10	11	12
Time (sec)	1000	450	300	230	280

teenth sensor estimation values. We average the time spent to do one entire process along 100 experiments.

As we can see in A.1, the computational cost is progressively decreasing from the eight sensor estimation, but then increases again from the twelfth. This shows that if we cut the algorithm when the fusion center is processing the first or last sensors approximations the amount of data is so large to be processed efficiently. We have to find a intermediate point where the exponentially increasing of information does not make our algorithm too slow.

The PSNR values will be showed deeply on 7, but the best updates points are $Update = [8, 10]$. So obviously, the choice will be $update = 10$ because the time spent in the process is much lower.