DISTRIBUTED ALGORITHMS FOR LOCALIZATION IN WIRELESS SENSOR NETWORKS

NICOLA PIOVESAN

Master's degree in Telecommunication Engineering
Department of Information Engineering
University of Padova

April 2016

Nicola Piovesan: *Distributed algorithms for localization in Wireless Sensor Networks,* Master's degree in Telecommunication Engineering, © April 2016

SUPERVISORS:
Prof. Tomaso Erseghe

LOCATION:
Padova

ABSTRACT

In this thesis work we introduce the concept of localization in Wireless Sensor Networks, starting from ranging measurements available at sensor nodes. Different measurements methods are explained.

We consider different solutions available in the literature and then we introduce the distributed localization algorithms proposed by two recent publications: the first solves the localization problem by optimizing a convex relaxation of the original problem, while the second solves the original non-convex problem in an ADMM fashion.

We initially exploit ADMM to speed up the convergence of the convex algorithm in terms of number of required iterations. Then, we propose an hybrid ADMM algorithm that initially considers the convex problem, exploiting its fast convergence and then switches to the original non-convex problem, in order to refine the achieved results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADMM Alternating Direction Method of Multipliers

CRLB Cramer Rao Lower Bound

GPS Global Positioning System

KKT Karush Kuhn Tucker

LOS Line of Sight

ML Maximum Likelihood

RMS Root Mean Square

RMSE Root Mean Square Error

RSS Received Signal Strength

RSSI Received Signal Strength Indicator

SDP Semi-Definite Programming

SNR Signal to Noise Ratio

TOA Time of Arrival

TOF Time of Flight

WSN Wireless Sensor Network

Part I

INTRODUCTION

1 INTRODUCTION

In recent years smart sensors became easiest to be developed thanks to the advent of new technologies, in particular thanks to the proliferation of Micro-Electro-Mechanical System (MEMS) technology and advances in RF communications. This kind of sensors are the basic unit of Wireless Sensor Networks (WSNs), which are extending our ability to monitor and control the physical world.

WSNs can be used either for monitoring and tracking applications. *Monitoring* include, for example, indoor/outdoor environmental monitoring, seismic monitoring, health monitoring, power monitoring, while *tracking* include tracking of objects, animals, humans and vehicles.

Sensor nodes can sense, measure and gather information from the environment and transmit these data to the user. They are characterized by their reduced dimension, limited processing and computing resources, and their low costs. There actually exist mechanical, biological, chemical, optical and magnetic sensors, which allow the deployment of densely distributed sensor networks that can be used for a very large range of monitoring applications, from marine to soil and atmospheric context, as we will explain in the next paragraph.

Sensors are also provided of a radio for wireless communications in order to allow them to transfer data to a base station which collects information from all the sensors. This provides to the user continuous and spatially dense observations on the environment.

The main source of power is the battery. Sensor can also be provided of a secondary power supply that harvests power from the environment. A classic example of this are solar panels. Sensors need to last for years or even decades without battery replacement; for this reason the consumption of power must be taken seriously in consideration.

A WSN usually has little or not infrastructure. It is composed of a number of sensor nodes (few tens to thousands) working together. We can distinguish two types of WSNs:

- *structured*: All or some of the nodes are deployed in a pre-planned manner, in this way their location is known.
- *unstructured*: Contains a dense collection of sensor nodes. Sensors may be deployed in an ad hoc[1] manner into the field.

The advantage of using structured networks is that fewer nodes are needed since their location can be chosen in order to optimize the

1 Sensor nodes can be placed randomly

coverage, while ad hoc deployments can have uncovered regions. Despite this, structured networks cannot always be deployed since we must consider in which specific environment sensors will be placed.

As we mentioned before, sensors have limited resources. This includes a limited amount of energy, short communication range, low bandwidth and limited processing and storage.

TYPES OF SENSOR NETWORKS There exists different kinds of sensor networks and their main characteristic is where they are deployed. They can typically be placed on land, underground and underwater. Depending on the environment a sensor network faces different challenges and resource constraints. There exists five types of WSN:

- *terrestrial*: It consists of hundreds to thousands nodes deployed either in an ad hoc or in a pre-planned manner. Terrestrial sensor nodes can be equipped with a secondary power source;
- *underground*: It consists of a number of sensor nodes placed underground or in a cave to monitor underground conditions. In this case, additional sink nodes are placed above ground to relay data from sensor nodes to the base station. The communication part is very important in this context because data transfer must be reliable through soil, rocks, water and other mineral contents that contribute to signal losses and high attenuation. Other important aspects are the deployment, that requires careful planning and energy and cost considerations. Energy is also important because sensors are equipped with a limited battery power and after the deployment, it is difficult to access to the sensor and then to recharge it.
- *underwater*: It consists of a number of sensor nodes deployed underwater. Usually these nodes communicate using acoustic waves and this comports limited bandwidth, long propagation delay and signal fading issue. Underwater sensor nodes, like in the underground case, are equipped with a limited battery which cannot be replaced or recharged.
- *multi-media*: This kind of sensor network enable monitoring and tracking of events in a multi-media form, such as video, audio and imaging. In this case, sensors are equipped with cameras and microphones and they must be deployed in a pre-planned manner in order to guarantee coverage.
- *mobile*: It consists of a number of sensor nodes that can move and interact with the physical environment. They have the ability to sense, compute and communicate like static nodes but they also have the ability to re position and organize themself in the network. Information gathered by a mobile node can be

communicated to another node when they are within a range of each other.

EXAMPLES OF APPLICATIONS Today WSNs are used in a large variety of applications. There exists smart structures that actively respond to earthquakes and make buildings safer. WSNs are employed in agriculture to reduce costs and environmental impact by watering and fertilizing only when necessary. Traffic monitoring systems use WSNs data to better control stoplights and alert drivers in case of traffic jams. Environmental monitoring networks can sense air, water and soil quality to identify the source of pollutants in real time. WSNs are also used in smart cities, for example for the parking slots management. Each parking slot owns a node of the network and the collected data can be used to identify unpaid parking tickets.

1.1 LOCALIZATION

An important challenge in WSNs is *localization*. In fact, it is fundamental to know the position of each sensor from which the base station is receiving data, in order to give sense to these data. The localization problem is yet extremely crucial for lots of applications since it can open up new ways of reducing power consumption. The first idea that can came into mind is to solve the problem by adding a Global Positioning System (GPS) receiver on the nodes. This is theoretically a feasible idea but for large networks of small, cheap and low power devices there are several considerations (cost, size and power usage) that preclude this way of solving the problem. Moreover, GPS receivers can be used only in outdoor applications, then this technology cannot be considered for indoor WSNs.

The localization problem aims at estimating the sensor nodes positions starting from ranging measurements of the distance between a node and its neighbors. Ranging can be performed using many approaches. The most common are Received Signal Strength Indicator (RSSI) and Time of Flight (TOF). The first represents a low-complexity technique and RSSI measures are today available in many wireless receivers. The TOF provides more reliable distances but the implementation of this technique is harder. In both cases the localization is done solving a Maximum Likelihood (ML) estimation problem, which is a non-convex problem of high dimensionality [10]. Some of the network nodes have known positions in order to guarantee a unique solution to the problem. These nodes are called *anchor nodes*.

1.1.1 *Centralized and distributed approaches*

There exists two main approaches to the resolution of this problem and we can resume them in the *centralized* and *distributed* paradigms.

The *centralized paradigm* expects that every node sends its information, in this case its ranging measurements, to a central processing unit that run the localization algorithm. If the nodes positions is required only at the central unit this is sufficient; in case each node must know its position, the central unit has to communicate to every node its computed position.

This kind of paradigm presents several problems. The main problem regards the traffic bottleneck since each node must send data to a unique central node. Furthermore, we must consider that this traffic grows while the network expands. This implies a scalability problem which affects the communication part as well as computation, since the localization problem solved by the central node becomes increasingly complex. Other highlighted problems concern the resilience to failure, the security and privacy issues that are typical when a centralized paradigm in considered.

The *distributed paradigm* is well suited for our case, where the problem is naturally distributed since we are dealing with nodes which are distributed in the space. We denotes with this paradigm an algorithm with no central units, where all nodes perform the same type of computations. This is the approach considered in this thesis work. In particular we can distinguish between two types of distributed WSN localization problem: the first type tries to solve the original non-convex problem. In some cases this approach can be dependent on the quality of the algorithm initialization. The second type of problem is a convex relaxation of the non-convex problem. The tightness of the convexification determines how close the solution to this problem approximates the original non-convex problem solution.

We must remind that these are sub-optimal methods that provide a solution which is not guaranteed to be global or local minimization point since the original and relaxed problem do not exactly coincide. Convex relaxation methods also increase the computational complexity of the problem [16].

In the past years a lot of approaches have been studied. Lot of them consider algorithms which are not guarantee to converge. An example of them are the linear approaches of [18] and [15], the multidimensional scaling approach of [6], the distributed gradient descent method of [9] and the spatially constrained local problem formulation of [8]. Other methods with convergence guarantee are based on convex relaxation techniques that cast the problem into a convex problem solved in a distributed fashion. The most used approach is the edge-based Semi-Definite Programming (SDP) relaxation introduced in [25] and [22], and the Alternating Direction Method of Multipliers (ADMM) approach covering convex relaxation [23].

A new relaxation approach is introduced in [24] in which the original ML problem is relaxed using the convex envelope of a distance function. This problem is then solved using a parallel distributed al-

gorithm with convergence guarantee. Another way to solve the localization problem is investigated in [11] where the proposed algorithm solves the original non-convex ML problem applying the ADMM paradigm in order to decompose the centralized non-linear problem in a number of non-linear local problems. Since ADMM guarantees convergence only in convex scenarios, the proposed algorithm is modified in such a way that penalty coefficients are locally increased in the presence of appropriate conditions.

In this thesis work we analyze the convex relaxation approach introduced in [24] and we show how it is possible to reduce the number of iterations required for the convergence of the algorithm using an ADMM approach. We follow investigating the approach introduced in [11], in which ADMM is applied to the original non-convex ML formulation. Then we provide a new method that consists in a combination of the convex and non-convex approaches. More precisely, we will use the non-convex method to refine the rough convex solution.

2 OVERVIEW ON MEASUREMENT METHODS

In the previous section we cited the concept of ranging measurements. In fact, each node must be able to provide to the algorithm a measurement that allows to estimate the distance to its neighbors. These measurements could be attained using different modalities, e.g., RF, infrared, acoustic or a combination of these.

An important discussion must be done on the quality of the measurements. In fact, range measurements used for localization are measured in a physical medium that introduces errors. We can distinguish between two kinds of errors:

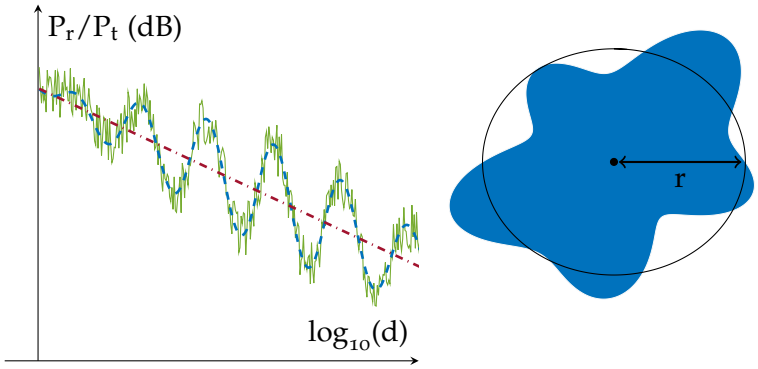
- *Time-varying errors*: The errors can be due to additive noise and interference. They can be reduced by averaging multiple measurements over time. This is a good way to mitigate this kind of errors because, usually, in localization applications, time delays are acceptable.
- *Environment-dependent errors*. They depend on the physical environment, specifically on the arrangement of the objects near the network (e.g., buildings, furniture, trees). In a network of stationary sensors, these errors are almost constant over time but since the environment is unpredictable, these errors are unpredictable too and they are, for this reason, modeled as random.

In the following sections we show how it is possible to obtain measurements that allow to estimate the nodes distances. In particular we examine the Received Signal Strength Indicator (RSSI) and Time of Flight (TOF) measurements.

2.1 RECEIVED SIGNAL STRENGTH INDICATOR

The RSSI is one of the most used approach to WSN localization. The Received Signal Strength (RSS) is defined as the voltage measured by a receiver's circuit. In some cases the RSS is reported as the power, that is the squared magnitude of the signal strength. Since the network nodes can communicate using several technologies, we can consider the RSS of RF, acoustic and other signals.

The RSS of RF signals can be measured by each node receiver during the normal data communication without requiring additional hardware or energy usage. This is a big advantage of this method, which is relatively inexpensive and simple to implement. The main disadvantage is that these measurements are notoriously unpredictable since they heavily depend on the transmission power (which is not



(a) Effects of path loss (red line), path loss and shadowing (blue line), path loss, shadowing and multipath (green line) on the SNR

(b) The real area of communication of a node is not the theoretical circle because of shadowing and multipath errors

Figure 1: Errors in RSSI measurements.

exactly known at the receiver) and the environment in which the network is deployed.

We know that in free space the power of a signal decays proportionally to the square of the distance between the transmitter and the receiver, as highlighted by the free space path loss equation

$$\mathrm{FSPL} = \left(\frac{4\pi d}{\lambda}\right)^2 \tag{1}$$

This is not true in real world channels where environment dependent errors increase the signal power reduction. These errors are caused by multipath signals and shadowing.

2.1.1 *Errors*

Multipath signals with different amplitudes and phases reach the receiver and they can sum destructively or constructively depending on the frequency. This fact causes frequency-selective fading. To diminish this effect could be used a spread-spectrum method that averages the received power over a wide range of frequencies.

The second source of errors that we listed is the shadowing. This is an attenuation of the signal due to obstructions (e.g, walls, buildings) between the transmitter and the receiver, that a signal must pass through or diffract around. Since the environment in which the WSN is deployed is unknown, the shadowing is modeled as random. In Figure 1 we can see how these source of errors affect the SNR and the communication area of nodes.

In the real world, the main received power decays proportionally to d^{-n_p}, where n_p is the path-loss exponent and its value is typically

in the range $2 < n_p < 4$. The mean power at distance d is frequently modeled as

$$\bar{P}(d) = P_0 - 10n_p \log \frac{d}{d_0} \tag{2}$$

where P_0 is the power (in dBm) received at a short reference distance d_0. The difference between the measured received power and its average, which corresponds to the shadowing contribution, is modeled as a log-normal [20]. We indicate with σ_{dB} the standard deviation of the received power (expressed in dBm). This value is relatively constant with distance and typically it assumes values in the range $4 \leqslant \sigma_{dB} \leqslant 12$. [21]. We indicate with $\mathbf{x}$ the coordinate vector which contains the positions of network nodes, defined as

$$\mathbf{x} = [x_1, \ldots, x_n, y_1, \ldots, y_n] \tag{3}$$

and with $d_{i,j}$ the distance between node i and node j computed as

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4}$$

The power received at node i (in dBm) and transmitted by node j, indicated as $P_{i,j}$, is distributed as

$$f(P_{i,j} = p|\mathbf{x}) = \mathcal{N}(p; \bar{P}(d_{i,j}), \sigma^2_{dB}) \tag{5}$$

which corresponds to the value at p of a Gaussian probability density function with mean $\bar{P}(d_{i,j})$ and variance , σ^2_{dB}. The log-normal model suggests that RSS-based range estimates have variance proportional to their actual range, and this is the reason why RSS errors are referred to as multiplicative, in comparison with the additive TOF errors.

2.2 TIME OF FLIGHT

The TOF method consists on using the amount of time required by the transmitted signal (RF, acoustic or other) to reach the receiver, to estimate the distance between the transmitter and the receiver. The time at which the receiver detects the signal corresponds to the time of transmission plus a propagation time delay. This time delay is quantified as

$$T_{i,j} = \frac{d_{i,j}}{v_p} \tag{6}$$

where v_p is the speed of the signal propagation, which for RF is approximately 10^6 times as fast as the speed of light. It is easy to see that from this equivalence we can compute the distance between node i and node j. We must take into consideration that additive noise and multipath signals hamper the receiver's ability to estimate the arrival time of the Line of Sight (LOS) signal.

2.2.1 *Errors*

We just saw that additive noise and multipath signals must be considered because they are an important source of errors. To estimate the Time of Arrival (TOA), which is the time at which the signal reaches the receiver, in a scenario where we have only additive noise, we can consider the time that maximize the cross-correlation between the received signals and the known transmitted signal. For a given bandwidth and SNR, the time delay estimate can only achieve a certain accuracy which is bound by the Cramer Rao Lower Bound (CRLB). If we consider a signal with bandwidth B with $B << F_c$, where F_c is the central frequency, with signal and noise power which are constant over the signal bandwidth, we can provide the bound

$$\text{var}\,(\text{TOA}) \geqslant \frac{1}{8\pi^2 B T_s F_c^2 \text{SNR}} \tag{7}$$

where T_s is the signal duration (in seconds). If the system is designed to achieve a sufficiently high SNR, the bound provided by (7) can be achieved in a multipath-free channel.

In multipath channels the errors can be many times greater than in the channel affected only by additive noise. This is due to the fact that all the late arriving multipath components are self-interference that decrease the SNR of the LOS signal. In this scenario, the receiver must find the first-arriving peak because it is not true that the LOS signal will be the strongest of the arriving signals. To do this we can use a threshold method in which we measure the first time that the cross-correlation crosses a pre-determined value.

Errors in TOA estimation are essentially caused by:

- *Early-arriving multipath*. In this case many multipath signals arrive right after the LOS signal and they contribute to the cross-correlation obscuring the location of the peak caused by the LOS signal. This problem typically causes small errors which are very difficult to combat.

- *Attenuated LOS*. The LOS signal is attenuated compared to the late-arriving multipath components. This causes the loss of the LOS signal, which is mistaken for noise. The attenuated LOS is a source of large errors but it is a severe problem only in low dense WSNs, in which the inter-sensor distance is high.

Finally, we must consider that errors in TOA estimations can also be due by delays in the transmitter and receiver hardware and software.

For short-range measurements, the measured time delay can be modeled with a Gaussian distribution

$$f\left(T_{i,j} = t|\mathbf{x}\right) = \mathcal{N}\left(t; \frac{d_{i,j}}{v_p} + \mu_T, \sigma_T^2\right) \tag{8}$$

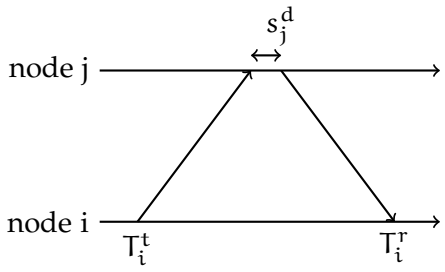


Figure 2: Two-way approach for sensors clocks synchronization.

where μ_T and σ_T^2 are the time delay mean and variance. As for the RSS case, $\mathbf{x}$ is the vector that contains the nodes positions. The mean error μ_T can be estimated by the localization algorithm and subsequently subtracted.

Finally, the presence of large errors make the tail of the measured TOA distribution heavier than a Gaussian, complicating the model. To solve this problem we can consider a mixture distribution.

Since the measurements of distance is based on the signal time of flight, it is particularly important to detect the exact value of $T_{i,j}$. To do this we need to have synchronized times at the transmitter and the receiver.

If the receiver clock is synchronized to the transmitter clock, the value of $T_{i,j}$ can easily be computed subtracting the known transmit time from the measured time of arrival. In sensor networks the algorithms for time synchronization reports precision of the order of $10\mu s$. This is a good precision for acoustic signals but not for RF signals. A typical solution to this problem is the *two-way TOA measurements approach*, shown in Figure 2. In this method, sensor i sends a signal to sensor j, which immediately replies with its own signal. So, sensor i can compute

$$T_{i,j} = \frac{T_i^r - T_i^t}{2} + s_j^d \tag{9}$$

where T^t is the time of transmission of the signal at node i, T^r is the time of arrival of the reply sent from sensor j, at node i and s_j^d is a reply delay internal to the sensor j. This value can be known or measured and sent to sensor i to be subtracted.

2.3 THE CRAMER RAO LOWER BOUND

The CRLB provides a way to calculate the lower bound on the covariance of any unbiased estimator that uses RSS or TOF. This is a good tool to design and to compare localization algorithms. The CRLB is a function of several WSN parameters, which are:

- number of nodes and number of anchor nodes;

- geometry of the sensor network;
- whether localization is in two or three dimensions;
- type of measurement implemented (i.e., RSS of TOF);
- channel parameters (σ_{dB} and n_p in RSS, σ_T in TOA);
- which pairs of sensors make measurements;
- unknown parameters that must be estimated (clock bias for TOF);

In the simulation results shown in Chapter 8 and in all the localization algorithms performance graphs shown in this work, we consider the CRLB in order to provide a simple way to judge the implemented algorithms.

2.3.1 *Computation of the CRLB*

In order to compute the CRLB we suppose to perform a two dimensional localization and to know the channel and device parameters (i.e., transmit powers and n_p for RSS measurements, clock biases for TOF). We proceed to the calculation of the CRLB for the estimate of the nodes position vector $\mathbf{x}$, defined in (3).

First, we need to form three $n \times n$ matrices, where n corresponds to the number of non-anchor nodes. These matrices are called $\mathbf{F}_{xx}$, $\mathbf{F}_{xy}$ and $\mathbf{F}_{yy}$. The k, l element of each matrix is calculated as:

$$
\begin{aligned}
[\mathbf{F}_{xx}]_{k,l} &= \begin{cases} \gamma \sum_{i \in H(k)} (x_k - x_i)^2 / d_{k,i}^s & k = l \\ -\gamma I_{H(k)}(l) (x_k - x_l)^2 / d_{k,l}^s & k \neq l \end{cases} \\
[\mathbf{F}_{xy}]_{k,l} &= \begin{cases} \gamma \sum_{i \in H(k)} (x_k - x_i)(y_k - y_i) / d_{k,i}^s & k = l \\ -\gamma I_{H(k)}(l) (x_k - x_l)(y_k - y_l) / d_{k,l}^s & k \neq l \end{cases} \\
[\mathbf{F}_{yy}]_{k,l} &= \begin{cases} \gamma \sum_{i \in H(k)} (y_k - y_i)^2 / d_{k,i}^s & k = l \\ -\gamma I_{H(k)}(l) (y_k - y_l)^2 / d_{k,l}^s & k \neq l \end{cases}
\end{aligned}
\tag{10}
$$

where $d_{i,j}$ is the real distance between node i and node j. The channel constant γ and the exponent s are functions of the measurements type and they are reported in Table 1. The indicator function $I_{H(k)}(l)$ is used to include the information only if sensor k made a measurement with sensor l, and it is defined as

$$
I_{H(k)} = \begin{cases} 1 & l \in H(k) \\ 0 & \text{otherwise} \end{cases} \tag{11}
$$

Then, we form the $2n \times 2n$ Fisher information matrix (FIM) corresponding to the $2n$ coordinates in $\mathbf{x}$ that need to be estimated

$$
\mathbf{F} = \begin{bmatrix} \mathbf{F}_{xx} & \mathbf{F}_{xy} \\ \mathbf{F}_{xy}^T & \mathbf{F}_{yy} \end{bmatrix} \tag{12}
$$

MEASUREMENT	CHANNEL CONSTANT γ	EXPONENT s
TOA	$\gamma = 1/\left(\nu_p \sigma_T\right)^2$	$s = 2$
RSS	$\gamma = \left(\frac{10 n_p}{\sigma_{dB} \log 10}\right)^2$	$s = 4$

Table 1: Parameters for CRLB computation

The CRLB matrix can be computed calculating the inverse matrix of $\mathbf{F}$. The diagonal of $\mathbf{F}^{-1}$ contains $2n$ values which are the variance bounds for the $2n$ parameters of $\mathbf{x}$.

2.4 SOME DERIVATION FROM THE CRLB COMPUTATION

It is interesting to look at how the variance bound changes by scaling the sensor network. We want to understand what happens when the geometry and connectivity of the network are kept constant, but the network dimensions are scaled up proportionally. We distinguish between the two examined measurements types:

- TOA: the CRLB will remain constant with a scaling of dimensions. This is due to the fact that $s = 2$ makes the fractions in (10) unit-less; the units come from the variance of the ranging error, $\nu_p \sigma_T$.
- RSS: CRLB is proportional to the network size. The equality $s = 4$ makes the ratios in (10) have the units of $1/\text{distance}^2$, so the variance bound takes its units of distance^2.

Obviously the channel parameters change slowly with the change of the path lengths but these scaling characteristics are a good approximation.

The bound on standard deviation of the localization error is also proportional to $(1/\gamma)^2$.

For the TOA case, it means that the error is proportional to the time delay standard deviation σ_T, which is an intuitive result. For the RSS case, it means that the localization error is proportional to the ratio σ_{dB}/n_p, so, if we are operating in a high path-loss exponent environment, we will have higher power usage but also a more accurate localization.

It is interesting to see how the CRLB changes while the nodes density increases. To do that we consider a sensor network located in a 20m $\times$ 20m square. This area contains L^2 sensors, L for each edge of the square, and the 4 of them which are in the corners are anchors nodes. An example is reported in Figure 3.

We consider both types of measurements introduced, with these parameters:

- RSS: $\sigma_{dB}/n_p = 1.7$

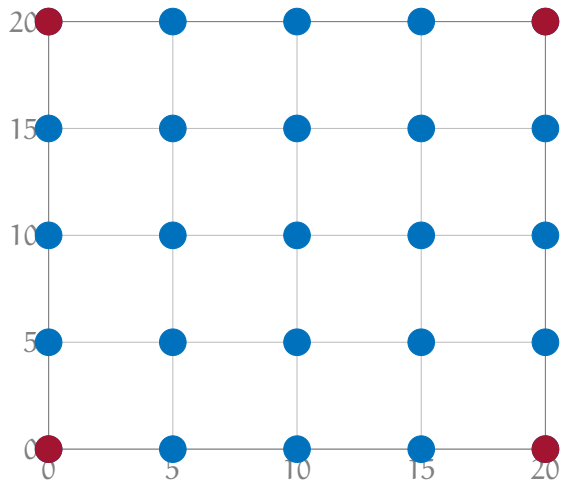


Figure 3: Example of network used in the computation of the CRLB for different densities cases. In this example a 25 nodes network ($L = 5$) is located in a 20m $\times$ 20m square.

- TOA: $\sigma_T = 6.1\text{ns}$, $v_p = 3 \cdot 10^8 \text{m/s}$

Since we have a lower bound for each node, we consider the Root Mean Square (RMS) value, which we calculate as

$$\text{RMS} = \sqrt{\frac{1}{L^2 - 4} \cdot \text{tr}\mathbf{F}^{-1}} \tag{13}$$

The RMS gives us an average of the bound over the entire $L^2 - 4$ nodes with unknown location. From Figure 4 we can see that the best lower bound is approached for $r = \infty$, which is the case in which each sensor can communicate with every sensor of the network. The other curves show the bound for the more realistic cases $r = 10\text{m}$ and $r = 15\text{m}$. For the chosen parameters we can see that in the realistic cases, TOA outperform RSS for low node densities but if the density increases RSS can perform as well as TOA. Obviously, this is true for the chosen parameters and the used network geometry. We also see that the RSS bound decreases more rapidly than TOA as the density increases.

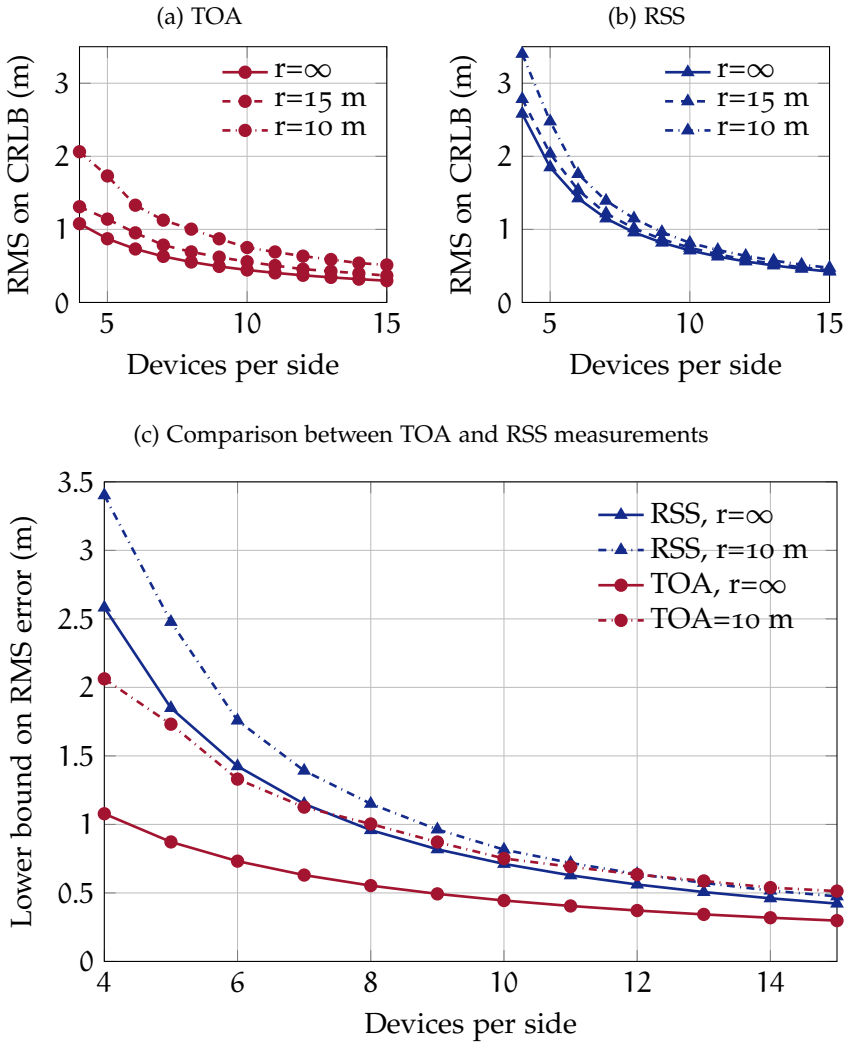


Figure 4: Lower bound on RMS of localization error using different measurement methods. The r parameter indicate the ray of communication of the node.

Part II

LOCALIZATION ALGORITHMS

3

CONVEX RELAXATION OF THE PROBLEM

The first important thing to do, before introducing different distributed algorithms, is to define the typical localization problem.

The Wireless Sensor Network (WSN) $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ is composed of N nodes with ranging capabilities. Each node is a member of the node set $\mathcal{N} = \{1, \cdots, N\}$, while $\mathcal{E}$ is the edge set.

The exact position of node i is indicated as $\mathbf{x}_i \in \mathbb{R}^n$ and it is a n-dimension value, where n represents the number of Cartesian coordinates ($n = 2$ for 2D localization, $n = 3$ for 3D localization). Neighbors of node i are denoted as $\mathcal{N}_i$ and we define a node $j, j \in \mathcal{N}_i$ as *neighbor* of node i if an estimate $r_{ij}, i \in \mathcal{N}, j \in \mathcal{N}_i$ of the true Euclidean distance $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ is available on both sides of the link.

The set of nodes with known position is $\mathcal{A}$ and these nodes are called *anchors*. The subset of anchors whose distance to node i is quantified by a noisy measurement, is $\mathcal{A}_i \subset \mathcal{A}$.

In this chapter we will provide a convex relaxation of the original localization problem, according to what recently proposed by Soares, Xavier, and Gomes [24] and then, we will numerical simulate the performance of this approach on three different sensor networks.

3.1 THE LOCALIZATION PROBLEM

As we mentioned before, we want to estimate the sensors positions $\mathbf{x} = \{\mathbf{x}_i : i \in \mathcal{N}\}$ from the available measurements $r_{ij}, i \sim j \in \mathcal{E}$ and from known anchors positions $\mathbf{x}_i = \mathbf{a}_i, i \in \mathcal{A}$. We assume that we are dealing with zero-mean, independent and identically distributed, additive Gaussian measurement noise.

The maximum likelihood estimator for the sensor positions is the solution of the optimization problem

$$\min_{\mathbf{x}} \sum_{i \sim j} \frac{1}{2} \left(\|\mathbf{x}_i - \mathbf{x}_j\| - r_{ij} \right)^2 + \sum_{i} \sum_{k \in \mathcal{A}_i} \frac{1}{2} \left(\|\mathbf{x}_i - \mathbf{a}_k\| - r_{ij} \right)^2 \quad (14)$$

where first term of (14) takes into consideration the ranging measurements between sensors, and the second term considers ranging measurements between sensors and anchors.

Problem (14) is non-convex and so, difficult to solve [23], nevertheless, it has a global minimum because the function we want to minimize is continuous and coercive [24].

It is useful to write this problem in another form, which will simplify the subsequent formulation of its convex relaxation.

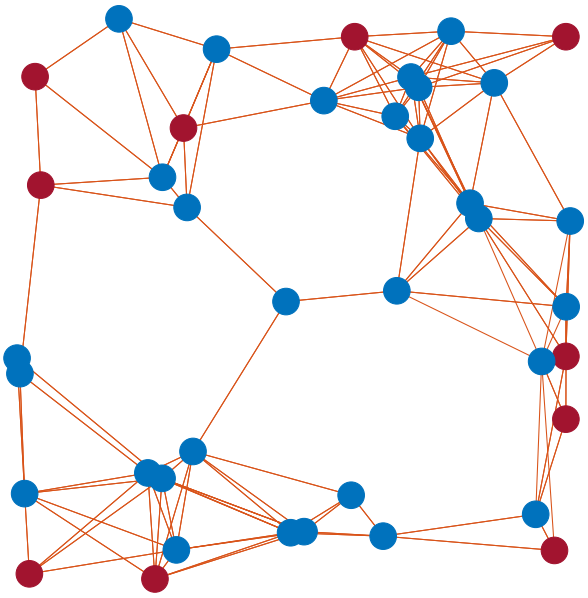


Figure 5: An example of WSN composed of 40 nodes. Anchor nodes are indicated in red. Edges indicate that the pair of nodes can communicate. This is one of the networks used in the later simulations.

We define $d^2_{\mathcal{C}}(\mathbf{z})$ the squared Euclidean distance of point $\mathbf{z}$ to set $\mathcal{C}$.

$$d^2_{\mathcal{C}}(\mathbf{z}) = \inf_{\mathbf{y}\in\mathcal{C}} \|\mathbf{z}-\mathbf{y}\|^2 \tag{15}$$

The spheres generated by the noisy measurements r_{ij} are represented by the sets

$$\begin{aligned} \mathcal{S}_{ij} &= \left\{\mathbf{z}\in\mathbb{R}^n : \|\mathbf{z}\| = r_{ij}\right\} \\ \mathcal{S}_{a_{ik}} &= \{\mathbf{z}\in\mathbb{R}^n : \|\mathbf{z}-\mathbf{a}_k\| = r_{ik}\} \end{aligned} \tag{16}$$

where the first set contains the points in the space $\mathbb{R}^n$ whose distance to the origin is equal to r_{ij}, while the second one takes into consideration the known positions of anchor nodes and contains the points in the space $\mathbb{R}^n$ whose distance to the k-anchor node is equal to r_{ik}.

We are, at this point, able to write the squared Euclidean distance of a generic point $\mathbf{z}$ to these set by using (15).

$$\begin{aligned} d^2_{\mathcal{S}_{ij}}(\mathbf{z}) &= \inf_{\|\mathbf{y}\|=r_{ij}} \|\mathbf{z}-\mathbf{y}\|^2 \\ d^2_{\mathcal{S}_{a_{ik}}}(\mathbf{z}) &= \inf_{\|\mathbf{y}-\mathbf{a}_k\|=r_{ik}} \|\mathbf{z}-\mathbf{y}\|^2 \end{aligned} \tag{17}$$

Taking into consideration functions (17), we can now rewrite problem (14) as

$$\min_{\mathbf{x}} \sum_{i\sim j} \frac{1}{2} d^2_{\mathcal{S}_{ij}}(\mathbf{x}_i - \mathbf{x}_j) + \sum_i \sum_{k\in\mathcal{A}_i} \frac{1}{2} d^2_{\mathcal{S}_{a_{ik}}}(\mathbf{x}_i) \tag{18}$$

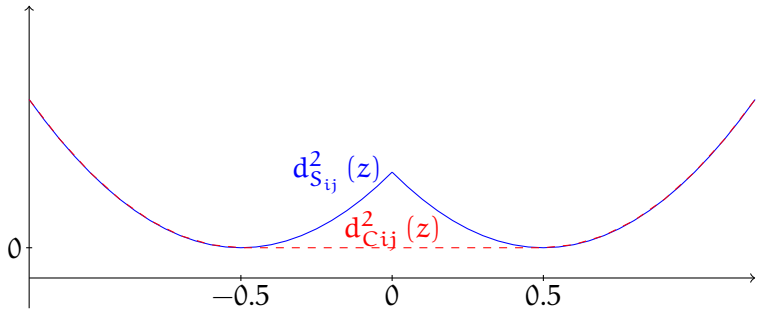


Figure 6: The squared distance to the ball $C_{ij} = \{z \in \mathbb{R} : |z| \leqslant 0.5\}$ (dotted line) is the convex envelope of the squared distance to the sphere $S_{ij} = \{z \in \mathbb{R} : |z| = 0.5\}$ (dashed line). The range measurement value, in this case, is $r_{ij} = 0.5$.

The non-convexity of this last problem follows from the non-convexity of functions (17) which we can consider the *building blocks* of this new formulation.

3.2 MAKE THE PROBLEM CONVEX

In the previous section we introduced the localization problem and we rewrote it as a function of squared Euclidean distance functions $d_{S_\cdot}(\mathbf{z})$. Now, we want to look for a way to find a convex problem which is similar to the original non-convex problem. In order to do this we will work on the non-convex functions (17).

We can proceed introducing the sets

$$\begin{aligned} \mathcal{C}_{ij} &= \left\{\mathbf{z} \in \mathbb{R}^n : \|\mathbf{z}\| \leqslant r_{ij}\right\} \\ \mathcal{C}_{a_{ik}} &= \{\mathbf{z} \in \mathbb{R}^n : \|\mathbf{z} - \mathbf{a}_k\| \leqslant r_{ik}\} \end{aligned} \tag{19}$$

which correspond to the convex hulls of sets (16). We can use these sets to define the *convex envelope* of the building blocks (17) as

$$\begin{aligned} d^2_{\mathcal{C}_{ij}}(\mathbf{z}) &= \inf_{\|\mathbf{y}\| \leqslant r_{ij}} \|\mathbf{z} - \mathbf{y}\|^2 \\ d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) &= \inf_{\|\mathbf{y} - \mathbf{a}_k\| \leqslant r_{ik}} \|\mathbf{z} - \mathbf{y}\|^2 \end{aligned} \tag{20}$$

The concept of convex envelope is illustrated in Figure 6 as a one-dimensional example. Its definition is pretty simple: if we consider a function γ, the *convex envelope* (or *convex hull*) of this function is its best possible convex under-estimator, $\operatorname{conv}\gamma(x) = \sup\{\eta(x) : \eta \leqslant \gamma, \eta \text{ is convex}\}$ and in general it is hard to determine.

These functions are the building blocks of the convex relaxation of Problem (18), which can be written as

$$\min_{\mathbf{x}} \hat{f}(\mathbf{x}) = \sum_{i \sim j} \frac{1}{2} d^2_{\mathcal{C}_{ij}}(\mathbf{x}_i - \mathbf{x}_j) + \sum_i \sum_{k \in \mathcal{A}_i} \frac{1}{2} d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) \tag{21}$$

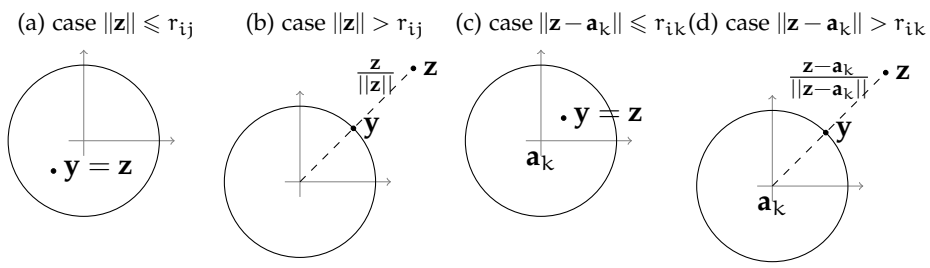


Figure 7: Examples of distinguished cases in analytical formulation of problem (22) (figures (a) and (b)) and problem (25) (figures (c) and (d)).

It is important to underline that the function in Problem (21) is an under-estimator of (18) but it is not the convex envelope of the original problem. Nevertheless, it can be demonstrated that in WSN localization applications it is a good approximation [24].

3.3 ANALYTICAL SOLUTION TO MINIMIZATION PROBLEM

We just derived a convex relaxation of problem (18). The building blocks of the optimization function of both problems (18) and (21) require the finding of infimum values. We can find these minimum values using numerical ways but this is poorly acceptable in terms of computation time of the final localization algorithm, so it is pretty smart, at this point, to find an analytical solution to these minimization problems.

Let's start with the first term of equation (21), which corresponds to a summation of $d^2_{\mathcal{C}_{ij}}$ functions, with

$$d^2_{\mathcal{C}_{ij}}(\mathbf{z}) = \inf_{\|\mathbf{y}\| \leqslant r_{ij}} \|\mathbf{z}-\mathbf{y}\|^2 \tag{22}$$

This function reaches its minimum when $\mathbf{y} = \mathbf{z}$. This solution is not always licit but we can say that the optimum solution corresponds to the closest allowed point $\mathbf{y}$ to $\mathbf{z}$. We distinguish between two cases:

- if $\|\mathbf{z}\| \leqslant r_{ij}$, then $\mathbf{z}$ is a member of the set from which we can choose $\mathbf{y}$, so the value that minimize the function is $\mathbf{y} = \mathbf{z}$ which allows the function to achieve its minimum possible value $d^2_{\mathcal{C}_{ij}} = 0$;
- if $\|\mathbf{z}\| > r_{ij}$ then the value of $\mathbf{y}$ that minimize the term is on the boundary of the circle $\|\mathbf{y}\| = r_{ij}$ and, more specifically, it corresponds to the intersection between the circle and the line that connect the point $\mathbf{z}$ to the origin, which is $\mathbf{y} = \frac{\mathbf{z}}{\|\mathbf{z}\|} r_{ij}$. With this value, the solution is $d^2_{\mathcal{C}_{ij}}(\mathbf{z}) = \left(\|\mathbf{z}\| - r_{ij}\right)^2$.

To summarize these results, we can write:

$$d^2_{\mathcal{C}_{ij}}(\mathbf{z}) = 1\left(\|\mathbf{z}\| - r_{ij}\right) \cdot \left(\|\mathbf{z}\| - r_{ij}\right)^2 \tag{23}$$

while the value of $\mathbf{y}$ that minimize the function is

$$\mathbf{y} = 1\left(\|\mathbf{z}\| - r_{ij}\right) \cdot \frac{\mathbf{z}}{\|\mathbf{z}\|} r_{ij} + 1\left(r_{ij} - \|\mathbf{z}\|\right) \cdot \mathbf{z} \tag{24}$$

where $1(\cdot)$ is the unit step function.

The second term of (21) makes use of the building block

$$d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) = \inf_{\|\mathbf{y}-\mathbf{a}_k\| \leqslant r_{ik}} \|\mathbf{z}-\mathbf{y}\|^2 \tag{25}$$

The only difference with the previous case is the set which defines the admissible $\mathbf{y}$ values. In this case the distance of point $\mathbf{y}$ from the known position of the anchor k must be lower or equal to r_{ik}. After considering the two separate cases as before, we find that this function can be re-written as

$$d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) = 1\left(\|\mathbf{z}-\mathbf{a}_k\| - r_{ik}\right) \cdot \left(\|\mathbf{z}-\mathbf{a}_k\| - r_{ik}\right)^2 \tag{26}$$

which corresponds to the solution given by

$$\mathbf{y} = 1\left(\|\mathbf{z}-\mathbf{a}_k\| - r_{ik}\right) \cdot \left(\mathbf{a}_k + \frac{\mathbf{z}-\mathbf{a}_k}{\|\mathbf{z}-\mathbf{a}_k\|} r_{ik}\right) + 1\left(r_{ik} - \|\mathbf{z}-\mathbf{a}_k\|\right) \cdot \mathbf{z} \tag{27}$$

3.4 DISTRIBUTED LOCALIZATION

We are at this point interested in the development of a synchronous distributed algorithm in which all nodes work in parallel, in order to find their locations. To do this, we need to compute the gradient of the cost function and its Lipschitz constant. We proceed to rewrite Problem (21) in an equivalent form, which allows to simplify the calculations.

$$\min_{\mathbf{x}} \hat{f} = \frac{1}{2} d^2_{\mathcal{C}}(\mathbf{A}\mathbf{x}) + \sum_i \sum_{k \in \mathcal{A}_i} \frac{1}{2} d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) \tag{28}$$

with $\mathbf{A} = \mathbf{M} \otimes \mathbf{I}_n$.

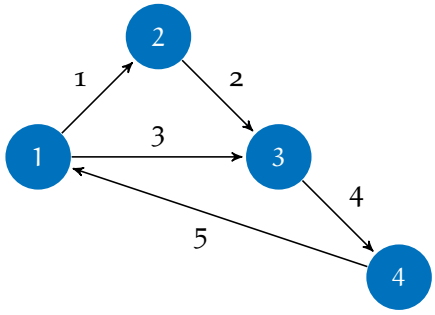
The symbol $\otimes$ indicates the Kronecker product, which is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \tag{29}$$

$\mathbf{M}$ is the arc-node incidence matrix of the network $\mathcal{G}$, $\mathbf{I}_n$ is the identity matrix of size n and $\mathcal{C}$ is the Cartesian product of the balls $\mathcal{C}_{ij}$ corresponding to all the edges of the network.

The algorithm requires that, during the setup, an arbitrary direction for each edge is set. This choice is stored in the $\mathbf{M}$ matrix, which is defined as

$$\mathbf{M}_{ij} = \begin{cases} 1 & \text{if arc } j \text{ starts at node } i \\ -1 & \text{if arc } j \text{ ends at node } i \\ 0 & \text{otherwise} \end{cases} \tag{30}$$



$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 1 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Figure 8: Example of network and its arc-node incidence matrix.

An example of arc-node incidence matrix is reported in Figure 8.

To simplify the calculation of the gradient and Lipschitz constant, we divide equation (28) into the summation of two functions

$$\min_{\mathbf{x}} \hat{f}(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x}) \tag{31}$$

where

$$g(\mathbf{x}) = \frac{1}{2} d^2_{\mathcal{C}}(\mathbf{A}\mathbf{x}), \qquad h(\mathbf{x}_i) = \sum_i (\mathbf{x}_i), \qquad h_i(\mathbf{x}) = \sum_{k \in \mathcal{A}_i} \frac{1}{2} d_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) \tag{32}$$

We also introduce the following functions to simplify the notation

$$\begin{aligned} \phi_{\mathcal{C}_{ij}}(\mathbf{z}) &= \frac{1}{2} d^2_{\mathcal{C}_{ij}}(\mathbf{z}) \\ \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) &= \frac{1}{2} d^2_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) \end{aligned} \tag{33}$$

3.4.1 *Proof of the problem equivalence*

We want to prove that the simplified Problem (28) is equivalent to Problem (21). Considering that the second term of (28) is a simple decomposition in summation of functions of the second term of equation (21), we only need to prove the equivalence of the first terms. The

product of matrix $\mathbf{M}$ with the $\mathbf{x}$ vector is the vector $\left(\mathbf{x}_i - \mathbf{x}_j : i \sim j\right)$. Function $g(\mathbf{x})$ can be written as

$$\begin{aligned} g(\mathbf{x}) &= \frac{1}{2} d^2_{\mathcal{C}}(\mathbf{A}\mathbf{x}) \\ &= \frac{1}{2} \inf_{y \in \mathcal{C}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2 \quad \text{(by definition (15))} \\ &= \frac{1}{2} \inf_{\|\mathbf{y}_{ij}\| \leqslant r_{ij}} \sum_{i \sim j} \|\mathbf{x}_i - \mathbf{x}_j - \mathbf{y}_{ij}\|^2 \\ &= \frac{1}{2} \sum_{i \sim j} \inf_{\|\mathbf{y}_{ij}\| \leqslant r_{ij}} \|\mathbf{x}_i - \mathbf{x}_j - \mathbf{y}_{ij}\|^2 \quad (*) \\ &= \sum_{i \sim j} \frac{1}{2} d^2_{\mathcal{C}_{ij}}\left(\mathbf{x}_i - \mathbf{x}_j\right) \quad \text{by definition (15)} \end{aligned} \tag{34}$$

where in step (*) we take in consideration the fact that all the terms are non-negative and the constraint set is a Cartesian product. This allows us to exchange inf with the summation.

This proves that the two problems are equivalent.

3.4.2 *Gradient and Lipschitz constant of g*

It can be demonstrated [14] that the function in (20), which we just defined as $\phi_{\mathcal{C}_{ij}}(\mathbf{z})$, is convex, differentiable and its gradient is

$$\nabla \phi_{\mathcal{C}_{ij}}(\mathbf{z}) = \mathbf{z} - P_{\mathcal{C}_{ij}}(\mathbf{z}) \tag{35}$$

where $P_{\mathcal{C}_{ij}}(\mathbf{z})$ is the orthogonal projection of $\mathbf{z}$ onto the closed convex set $\mathcal{C}_{ij}$, which we can write as

$$P_{\mathcal{C}_{ij}}(\mathbf{z}) = \arg\min_{\mathbf{y} \in \mathcal{C}_{ij}} \|\mathbf{z} - \mathbf{y}\| \tag{36}$$

It can also be demonstrated [24] that function $\phi_{\mathcal{C}_{ij}}$ has a Lipschitz continuous gradient with constant $L_\phi = 1$.

We define the function $\phi_{\mathcal{C}}$ which is the result of summing all the functions $\phi_{\mathcal{C}_{ij}}$. This function allows us to write $g(\mathbf{x}) = \phi_{\mathcal{C}}(\mathbf{A}\mathbf{x})$. Considering (35), we can write the gradient of the g function as

$$\begin{aligned} \nabla g(\mathbf{x}) &= \mathbf{A}^T \nabla \phi_{\mathcal{C}}(\mathbf{A}\mathbf{x}) \\ &= \mathbf{A}^T \left(\mathbf{A}\mathbf{x} - P_{\mathcal{C}}(\mathbf{A}\mathbf{x})\right) \\ &= \boldsymbol{\mathcal{L}}\mathbf{x} - \mathbf{A}^T P_{\mathcal{C}}(\mathbf{A}\mathbf{x}) \end{aligned} \tag{37}$$

where $\boldsymbol{\mathcal{L}} = \mathbf{A}^T\mathbf{A} = \mathbf{L} \otimes \mathbf{I}_n$, and $\mathbf{L}$ is the Laplacian matrix of the graph. This gradient is Lipschitz continuous and we can proceed with calculations to find the Lipschitz constant.

$$\begin{aligned}
\|\nabla g(\mathbf{x}) - \nabla g(\mathbf{y})\| &= \|\mathbf{A}^T(\nabla \phi_{\mathcal{C}}(\mathbf{A}\mathbf{x}) - \nabla \phi_{\mathcal{C}}(\mathbf{A}\mathbf{y}))\| \\
&\leqslant \|\mathbf{A}\|\,\|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y}\| \\
&\leqslant \|\mathbf{A}\|^2\|\mathbf{x} - \mathbf{y}\| \\
&= \lambda_{\max}\left(\mathbf{A}^T\mathbf{A}\right)\|\mathbf{x} - \mathbf{y}\| \\
&= \lambda_{\max}(L)\,\|\mathbf{x} - \mathbf{y}\| \quad \text{(Kronecker product property)} \\
&\leqslant 2\delta_{\max}\|\mathbf{x} - \mathbf{y}\| \quad [7]
\end{aligned} \tag{38}$$

where $\|\mathbf{A}\|$ is the maximum singular value norm and $\delta_{\max}$ is the maximum node degree of the network $\mathcal{G}$. From the definition of Lipschitz constant we can recognize that $L_g = 2\delta_{\max}$.

3.4.3 *Gradient and Lipschitz constant of h*

We defined the h function as a summation of h_i functions so the gradient of $h(x)$ is $\nabla h(\mathbf{x}) = (\nabla h_1(\mathbf{x}_1), \ldots, \nabla h_n(\mathbf{x}_n))$. From the definition of h_i as summation of $\phi_{\mathcal{C}_{a_{ik}}}$ functions, we can write

$$\begin{aligned}
\nabla h_i(\mathbf{x}_i) &= \sum_{k \in \mathcal{A}_i} \nabla \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) \\
&= \sum_{k \in \mathcal{A}_i} \mathbf{x}_i - P_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i)
\end{aligned} \tag{39}$$

The next step is calculating the Lipschitz constant for each h_i.

$$\begin{aligned}
\|\nabla h_i(\mathbf{x}_i) - \nabla h_i(\mathbf{y}_i)\| &\leqslant \left\| \sum_{k \in \mathcal{A}_i} \nabla \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) - \sum_{k \in \mathcal{A}_i} \nabla \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{y}_i) \right\| \\
&\leqslant \sum_{k \in \mathcal{A}_i} \left\| \nabla \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{x}_i) - \nabla \phi_{\mathcal{C}_{a_{ik}}}(\mathbf{y}_i) \right\| \\
&\leqslant |\mathcal{A}_i|\,\|\mathbf{x}_i - \mathbf{y}_i\|
\end{aligned} \tag{40}$$

From this result we can see that it is $L_{h_i} = |\mathcal{A}_i|$, so the Lipschitz constant of the h_i function correspond to the cardinality of the set of anchors which are neighbors of node i. This allows us to calculate the overall Lipschitz constant

$$\begin{aligned}
\|\nabla h(\mathbf{x}) - \nabla h(\mathbf{y})\| &= \sqrt{\sum_i \|\nabla h_i(\mathbf{x}_i) - \nabla h_i(\mathbf{y}_i)\|^2} \\
&\leqslant \sqrt{\sum_i |\mathcal{A}_i|^2\,\|\mathbf{x}_i - \mathbf{y}_i\|^2} \\
&\leqslant \max(|\mathcal{A}_i| : i \in \mathcal{N}) \cdot \|\mathbf{x} - \mathbf{y}\|
\end{aligned} \tag{41}$$

that is $L_h = \max(|\mathcal{A}_i| : i \in \mathcal{N})$.

Now we know everything is necessary to calculate what we really need, namely the gradient and the Lipschitz constant of function $\hat{f}$. The gradient of the minimization problem objective function $\hat{f}$ can be computed by summing the gradient of the g and the h functions.

$$\begin{aligned}\nabla\hat{f}(\mathbf{x}) &= \nabla g(\mathbf{x}) + \nabla h(\mathbf{x}) \\ &= \boldsymbol{\mathcal{L}}\mathbf{x} - \mathbf{A}^T P_{\mathcal{C}}(\mathbf{A}\mathbf{x}) + \begin{bmatrix} \sum_{k\in\mathcal{A}_1} \mathbf{x}_1 - P_{\mathcal{C}_{a_{1k}}}(\mathbf{x}_1) \\ \vdots \\ \sum_{k\in\mathcal{A}_n} \mathbf{x}_n - P_{\mathcal{C}_{a_{nk}}}(\mathbf{x}_n) \end{bmatrix}\end{aligned} \tag{42}$$

In the same way, the Lipschitz constant of $\hat{f}$ can be obtained by summing the Lipschitz constants of functions h and g.

$$\begin{aligned}L_{\hat{f}} &= L_g + L_h \\ &= 2\delta_{max} + \max(|\mathcal{A}_i| : i \in \mathcal{N})\end{aligned} \tag{43}$$

3.4.4 *Analytical solution to orthogonal projection functions*

The minimization problems introduced by the use of orthogonal projections can be analytically solved in a very similar way to what we showed in Section 3.3. The following equations show how the projection of a $\mathbf{z}$ point to sets $\mathcal{C}_{ij}$ or $\mathcal{C}_{a_{ik}}$, which corresponds to a constrained minimization problem, can be computed in an analytical way.

$$\begin{aligned}P_{\mathcal{C}_{ij}}(\mathbf{z}) &= \arg\min_{\mathbf{y}\in\mathcal{C}_{ij}} \|\mathbf{z}-\mathbf{y}\| = \left(\frac{\mathbf{z}}{\|\mathbf{z}\|} r_{ij}\right) 1\left(\|\mathbf{z}\| - r_{ij}\right) + \mathbf{z}\cdot 1\left(r_{ij} - \|\mathbf{z}\|\right) \\ P_{\mathcal{C}_{a_{ik}}}(\mathbf{z}) &= \arg\min_{\mathbf{y}\in\mathcal{C}_{a_{ik}}} \|\mathbf{z}-\mathbf{y}\| = \left(\frac{\mathbf{z}-\mathbf{a}_k}{\|\mathbf{z}-\mathbf{a}_k\|} r_{ik} + \mathbf{a}_k\right) 1\left(\|\mathbf{z}-\mathbf{a}_k\| - r_{ik}\right) \\ &\qquad + \mathbf{z}\cdot 1\left(r_{ik} - \|\mathbf{z}-\mathbf{a}_k\|\right)\end{aligned} \tag{44}$$

3.5 DISTRIBUTED PARALLEL ALGORITHM

The distributed parallel algorithm can be implemented following the Nesterov's optimal method [19] thanks to the Lipschitz continuity of the gradient of $\hat{f}$.

We initially consider the first term of (42), that we previously named $\nabla g(\mathbf{x})$. The i-th entry of $\boldsymbol{\mathcal{L}}\mathbf{x}$ can be computed by node i considering estimates of its own positions and its neighbors positions. It is $\boldsymbol{\mathcal{L}}\mathbf{x} = \delta_i \mathbf{x}_i - \sum_{j\in\mathcal{N}_i} \mathbf{x}_j$, where δ_i is the degree of node i.

In the second part we have $\mathbf{A}^T P_{\mathcal{C}}(\mathbf{A}\mathbf{x})$. We can write $P_{\mathcal{C}}(\mathbf{A}\mathbf{x}) = \{P_{\mathcal{C}_{ij}}(\mathbf{x}_i - \mathbf{x}_j)\}_{i\sim j\in\mathcal{E}}$. Each term of this projection depends only on the

edge terminals and the noisy measurements r_{ij}. The product with $\mathbf{A}^T$ collects, for each node, the sum of the projections relative to edges where it intervenes, with positive or negative signs depending on the arbitrary edge directions. Summing up we can write

$$\left(\mathbf{A}^T P_{\mathcal{C}}\left(\mathbf{A}\mathbf{x}\right)\right)_i = \sum_{j \in \mathcal{N}_i} c_{i\sim j,i} P_{\mathcal{C}_{ij}}\left(\mathbf{x}_i - \mathbf{x}_j\right) \tag{45}$$

where $c_{(i\sim j,i)}$ denotes the entry $(i \sim j, i)$ in the arc-node incidence matrix $\mathbf{M}$.

The second term of equation (42) can be easily implemented using equation (39).

Finally, the position update requires the computation of the gradient of the cost with respect to the coordinates of node i, evaluated at the extrapolated point $\mathbf{w}_i$, where this value can be previously calculated as $\mathbf{w}_i = \mathbf{x}_i(k-1) + \frac{k-2}{k+1}\left(\mathbf{x}_i(k-1) - \mathbf{x}_i(k-2)\right)$, using a standard application of Nesterov's method [24].

The final parallel distributed algorithm is reported in Algorithmus 1. The required inputs are the noisy measurements between neighbors nodes which are indicated as r_{ij}, in the case we are dealing with non-anchor nodes, and with r_{ik} in the case one of the nodes in an anchor. The other required input value is the Lipschitz constant $L_{\hat{f}}$ which depends on the geometry of the network, in particular on the maximum node degree and on the maximum number of anchors that can be neighbors of a single node. This value can be computed using equation (43).

Input : $L_{\hat{f}}$, $\{r_{ij} | i \sim j\}$,$\{r_{ik} | k \in \mathcal{A}\}$
Output : $\hat{\mathbf{x}}$
$k = 0$;
each node i chooses random $\mathbf{x}_i(0) = \mathbf{x}_i(-1)$;
while *some stop criterion is not met, each node* i **do**
  $k = k + 1$;
  $\mathbf{w}_i = \mathbf{x}_i(k-1) + \frac{k-2}{k+1}\left(\mathbf{x}_i(k-1) - \mathbf{x}_i(k-2)\right)$
  node i broadcast $\mathbf{w}_i$ to its neighbors;
  $\nabla g_i(\mathbf{w}_i) = \delta_i \mathbf{w}_i - \sum_{j\in N_i} \mathbf{w}_j + \sum_{j\in N_i} c_{(i\sim j,i)} P_{C_{ij}}\left(\mathbf{w}_i - \mathbf{w}_j\right)$
  $\nabla h_i(\mathbf{w}_i) = \sum_{k\in\mathcal{A}_i} \mathbf{w}_i - P_{C_{a_{ik}}}(\mathbf{w}_i)$
  $\mathbf{x}_i(k) = \mathbf{w}_i - \frac{1}{L_{\hat{f}}}\left(\nabla g_i(\mathbf{w}_i) + \nabla h_i(\mathbf{w}_i)\right)$
end
return $\hat{\mathbf{x}} = \mathbf{x}(k)$

Algorithmus 1 : Parallel algorithm

3.6 NUMERICAL SIMULATIONS

We want to examine the performance of this algorithm on different networks. In particular, we will deal with three different networks.

The accuracy of the algorithms is measured by the Root Mean Square Error (RMSE) per sensor, which we define as

$$\text{RMSE} = \frac{\mathrm{E}\left[\left|\left|\mathbf{x}^t - \mathbf{x}\right|\right|\right]}{\sqrt{N}} \tag{46}$$

where $\mathbf{x}^t$ are the computed nodes positions at iteration t, $\mathbf{x}$ are the real locations of network nodes, N is the total number of nodes and the average E takes into account the mean over the different Monte Carlo realizations. The noisy range measurements are generated according to

$$r_{ij} = d_{ij} + \nu_{ij} \tag{47}$$

where d_{ij} is the true Euclidean distance between node i and j, while $\{\nu_{ij} : i \in \mathcal{N}, j \in \mathcal{N}_i\}$ are independent Gaussian random variables with zero mean and standard deviation σ.

The first network consists of 40 nodes and it corresponds to the example network shown in Figure 5. Nodes are distributed on a square of unit side and 10 of them are anchor nodes (they are indicated by red circles). The edges shown in the figure represent the available ranging measurements and it is guaranteed that at least three measurements with neighbors are available to each node, in order to avoid multiple solutions to the optimization problem. The noise level is $\sigma = 10^{-1}$. In Figure 9a we can see the path followed by values $\mathbf{x}^t$ as iteration t progress. All paths start from the $\mathbf{x}^0 = \mathbf{0}$ point, and the crosses indicate the nodes positions at $t = 200$. We note that for some nodes the computed locations are very close to the real positions indicated by the circles. The RMSE evolution over the number of iterations performed by the algorithm is reported in Figure 9b

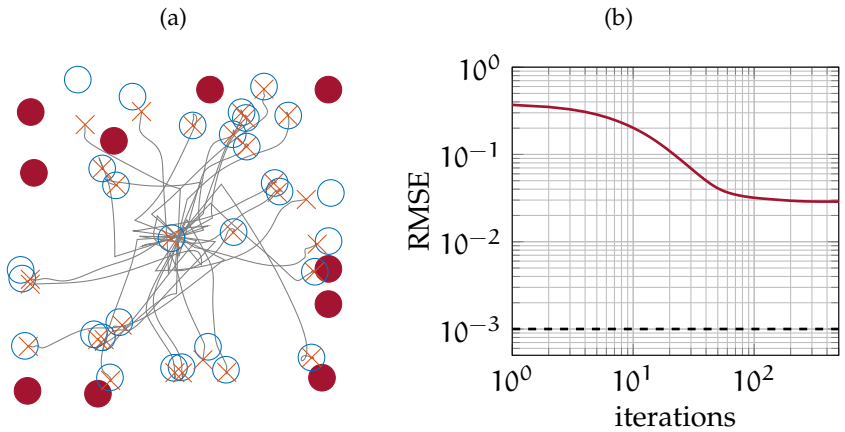


Figure 9: Performances of the distributed parallel algorithm for the N=40 nodes network. (a) Convergence path. Circles indicate the correct positions of network nodes, while crosses indicate positions $\mathbf{x}^t$ at iteration $t = 200$. (b) RMSE performance.

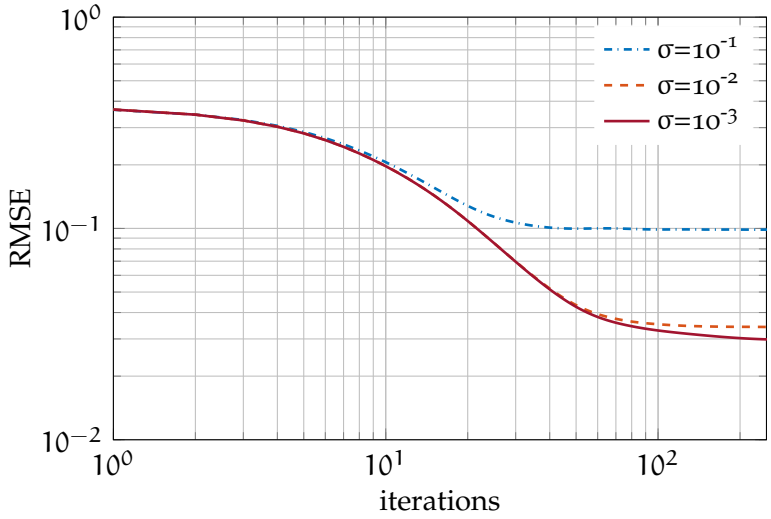


Figure 10: Average RMSE performance measure for the N = 40 nodes network versus the iteration number.

The second test is performed on the 40-nodes network, but in this case we average the RMSE over 50 noise realizations for each noise level. The considered noise standard deviations are $\sigma = 10^{-1}$, $\sigma = 10^{-2}$ and $\sigma = 10^{-3}$. In Figure 10, the algorithm performances for the three noise levels are shown. In particular we note that there are small differences in the RMSE values at convergence between the $\sigma = 10^{-2}$ and $\sigma = 10^{-3}$ cases. The algorithm, for $\sigma = 10^{-1}$ noise level, converges faster but to an higher RMSE value.

The last test is performed on larger networks, provided as benchmark tests for localization algorithms by [26]. The first one is composed of 500 node, of which 10 are anchor nodes, while the second is composed of 1000 nodes, of which 20 are anchors. These networks are a very useful tool to test algorithms performance because the large dimension guarantee the presence of lots of nodes configuration cases that may make the localization difficult. In Figure 11a and Figure 11b the RMSE evolution over the iterations number is shown for both 500 nodes and 1000 nodes cases. Considering the RMSE performance of all the testing networks, we note that the convergence is faster for smaller network.

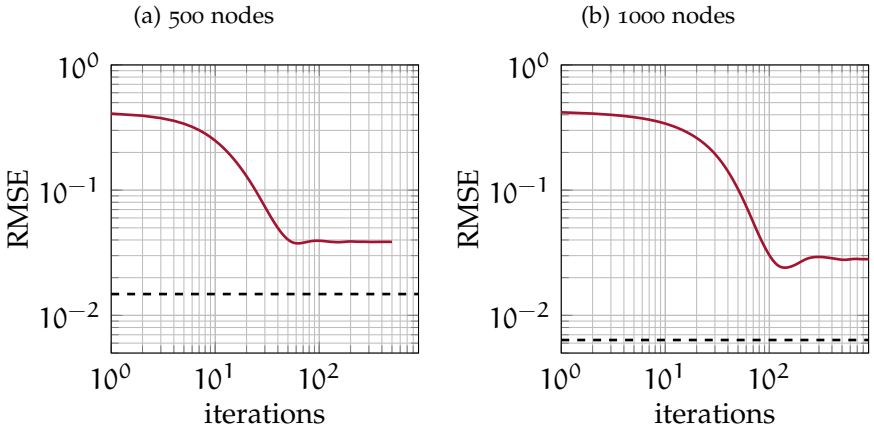


Figure 11: RMSE performance measure for the N = 500 and N = 1000 nodes network versus the iteration number.

4

ALTERNATING DIRECTION METHOD OF MULTIPLIERS

The Alternating Direction Method of Multipliers (ADMM) is a simple but powerful algorithm that solves optimization problems decomposing them into smaller local sub-problems, which are easier to handle. The solutions to these local subproblems are coordinated to find the solution to a global problem. This algorithm is well suited for distributed optimization and in the latest years it found several applications in different areas.

It was first introduced in the mid 1970s in the works [12] and [13] but similar ideas emerged as early as the mid-1950 [5]. Today this approach finds new interests thanks to the presence of large-scale distributed computing systems and the needs to solve massive optimization problems.

There exists several formulations of the ADMM algorithm. In this work we refer to the formulation presented in the Bertsekas book [3], also used in [11].

In this chapter we want to exploit the ADMM to improve the performances of the algorithm introduced in Chapter 3, in particular to reduce the number of iterations necessary to the algorithm to converge.

We underline that the parallel distributed algorithm seen in Chapter 3, requires the exchange of information at each iteration. More in detail, every node has to broadcast the $\mathbf{w}_i$ vector to its neighbors. This means that, reducing the total number of iterations performed by the algorithm implies a lower number of communications between nodes and so a less usage of energy.

In order to introduce the algorithm in a generic way, which will be useful for further applications with different optimization functions, we will perform all the calculations and derivations starting from a generic problem, that we indicate as

$$\begin{aligned} &\min \sum_{i\sim j} f_{i,j}\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right) \\ &\text{w.r.t. } \mathbf{x}_i,\ i \in \mathcal{N} \\ &\text{s.t. } \mathbf{x}_k = \mathbf{a}_k,\ \forall k \in \mathcal{A} \end{aligned} \tag{48}$$

We need to put this problem in a form which is suitable for the application of the ADMM. To do this, we duplicate the $\mathbf{x}_i$ variables. In this way, the generic node i owns its copy of variables $\mathbf{x}_j$ which is represented by $\hat{\mathbf{x}}_{i,j}, j \in \mathcal{N}_i \cup \{i\}$. In particular, the first element of this

vector contains the position of node i, $\hat{\mathbf{x}}_{i,i} = \mathbf{x}_i$, while the remaining ones represent its neighbors positions $\hat{\mathbf{x}}_{i,j} = \mathbf{x}_j, j \in \mathcal{N}_i$.

The previous problem can be rewritten into a new form taking in consideration these duplicated variables.

$$\begin{aligned} &\min \sum_{i\sim j} f_{i,j}\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\|\right) \\ &\text{w.r.t } \hat{\mathbf{x}}_{i,j},\ i \in \mathcal{N},\ j \in \mathcal{N}_i \cup \{i\} \\ &\text{s.t. } \hat{\mathbf{x}}_{j,i} = \hat{\mathbf{x}}_{i,i},\ \forall i \in \mathcal{N},\ j \in \mathcal{N}_i \\ &\qquad \hat{\mathbf{x}}_{k,k} = \mathbf{a}_k,\ \forall k \in \mathcal{A} \end{aligned} \tag{49}$$

where $\hat{\mathbf{x}}_{j,i}$ is the replica of variable $\mathbf{x}_i$ available at node j.

The formulation used in Problem (49) is probably the most intuitive form of replication for variables $\hat{\mathbf{x}}_{i,j}$ but yet not the most appropriate in terms of performance.

An alternative form, which is better suited for the application of ADMM, consists on rewriting the equivalence $\hat{\mathbf{x}}_{j,i} = \hat{\mathbf{x}}_{i,i}$ as

$$\begin{aligned} \hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j} &= \hat{\mathbf{x}}_{j,i} - \hat{\mathbf{x}}_{j,j} \\ \hat{\mathbf{x}}_{i,i} + \hat{\mathbf{x}}_{i,j} &= \hat{\mathbf{x}}_{j,i} + \hat{\mathbf{x}}_{j,j} \end{aligned} \tag{50}$$

which corresponds to the constraints $\hat{\mathbf{x}}_{j,i} = \hat{\mathbf{x}}_{i,i}$ and $\hat{\mathbf{x}}_{i,j} = \hat{\mathbf{x}}_{j,j}$.

With this new representation of the equivalence we can rewrite the minimization problem into a new form.

$$\begin{aligned} &\min \sum_{i\sim j} f_{i,j}\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\|\right) \\ &\text{w.r.t. } \hat{\mathbf{x}}_{i,i}, \hat{\mathbf{x}}_{i,j}, \mathbf{z}_{1,i,j}, \mathbf{z}_{2,i,j},\ i \in \mathcal{N}, j \in \mathcal{N}_i \\ &\text{s.t. } \hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j} = \mathbf{z}_{1,i,j},\ \forall i \in \mathcal{N}, j \in \mathcal{N}_i \\ &\qquad \hat{\mathbf{x}}_{i,i} + \hat{\mathbf{x}}_{i,j} = \mathbf{z}_{2,i,j},\ \forall i \in \mathcal{N}, j \in \mathcal{N}_i \\ &\qquad \mathbf{z}_{1,i,j} = -\mathbf{z}_{1,j,i},\ \forall i \in \mathcal{N}, j \in \mathcal{N}_i \\ &\qquad \mathbf{z}_{2,i,j} = \mathbf{z}_{2,j,i},\ \forall i \in \mathcal{N}, j \in \mathcal{N}_i \\ &\qquad \mathbf{x}_{k,k} = \mathbf{a}_k,\ \forall k \in \mathcal{A} \end{aligned} \tag{51}$$

In this new formulation of the ADMM problem, we used two auxiliary variables, $\mathbf{z}_{1,i,j}$ and $\mathbf{z}_{2,i,j}$, which needs to belong, respectively, to linear spaces $\mathcal{Z}_1 = \left\{\mathbf{z}_{1,i,j} = -\mathbf{z}_{1,j,i}\right\}$ and $\mathcal{Z}_2 = \left\{\mathbf{z}_{2,i,j} = \mathbf{z}_{2,j,i}\right\}$, in order to force equivalence (50).

This is the definitive problem formulation that we will use but, in order to develop the ADMM algorithm, it may be useful to rewrite it in a compact form which will allow to simplify the algorithm code, its performance and, nevertheless, the understandability of this solution.

4.1 COMPACT FORM OF THE PROBLEM

First, we can group together all variables used by the generic node i. They are the position vectors $\hat{\mathbf{x}}_{i,j}$ and the auxiliary variables $\mathbf{z}_{1,i,j}$, $\mathbf{z}_{2,i,j}$, that can be grouped in

$$\hat{\mathbf{x}}_i = \begin{bmatrix} \hat{\mathbf{x}}_{i,i} \\ \left[\hat{\mathbf{x}}_{i,j}\right]_{j\in\mathcal{N}_i} \end{bmatrix}, \qquad \mathbf{z}_i = \begin{bmatrix} \left[\mathbf{z}_{1,i,i}\right]_{j\in\mathcal{N}_i} \\ \left[\mathbf{z}_{2,i,j}\right]_{j\in\mathcal{N}_i} \end{bmatrix} \tag{52}$$

Each node of the network owns a $\hat{\mathbf{x}}_i$ and a $\mathbf{z}_i$ vector. All of them can be collected in the global vectors $\hat{\mathbf{x}} = [\hat{\mathbf{x}}_i]_{i\in\mathcal{N}}$ and $\mathbf{z} = [\mathbf{z}_i]_{i\in\mathcal{N}}$.

With these new vectors definitions the reference problem becomes

$$\begin{aligned} &\min \sum_{i\in\mathcal{N}} F_i\left(\hat{\mathbf{x}}_i\right) \\ &\text{w.r.t. } \hat{\mathbf{x}} \in \mathcal{X}, \mathbf{z} \in \mathcal{Z} \\ &\text{s.t. } \mathbf{A}_i\hat{\mathbf{x}}_i = \mathbf{z}_i, \forall i \in \mathcal{N} \end{aligned} \tag{53}$$

where F_i are the local cost functions, which are defined as

$$F_i\left(\hat{\mathbf{x}}_i\right) = \sum_{j\in\mathcal{N}_i} f_{i,j}\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\|\right) \tag{54}$$

Variables $\hat{\mathbf{x}}$ and $\mathbf{z}$ belong, respectively, to linear spaces

$$\begin{aligned} &\mathcal{X} = \{\hat{\mathbf{x}} | \hat{\mathbf{x}}_{k,k} = \mathbf{a}_k, \forall k \in \mathcal{A}\} \\ &\mathcal{Z} = \left\{\mathbf{z} | \mathbf{z}_{1,i,j} = -\mathbf{z}_{1,j,i}, \mathbf{z}_{2,i,j} = \mathbf{z}_{2,j,i}, \forall i \in \mathcal{N}, j \in \mathcal{N}_i\right\} \end{aligned} \tag{55}$$

Matrices $\mathbf{A}_i$ are full rank and defined as

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{1}_{N_i} & -\mathbf{I}_{N_i} \\ \mathbf{1}_{N_i} & \mathbf{I}_{N_i} \end{bmatrix} \otimes \mathbf{I}_n \tag{56}$$

where $N_i = |\mathcal{N}_i|$ is the neighbors cardinality, $\mathbf{I}_{N_i}$ an identity matrix of size N_i and $\mathbf{1}_{N_i}$ a column vector of length N_i with all entries equal to one. The Kronecker product with the $\mathbf{I}_n$ matrix is used to duplicate the values for each dimension of the total considered n dimensions.

4.2 DISTRIBUTED ADMM ALGORITHM

There exists several versions of the ADMM algorithm, which, moreover, is the most used method for distributed coordination of agents [3] [5]. In this section we will refer to the formulation of Bertsekas [3]. We start by writing the optimization problem as

$$\begin{aligned} &\min F\left(\hat{\mathbf{x}}\right) \\ &\text{w.r.t. } \hat{\mathbf{x}} \in \mathcal{X}, \mathbf{z} \in \mathcal{Z} \\ &\text{s.t. } \mathbf{A}\hat{\mathbf{x}} = \mathbf{z} \end{aligned} \tag{57}$$

where the object function $F(\hat{\mathbf{x}}) = \sum_{i \in \mathcal{N}} F_i(\hat{\mathbf{x}}_i)$ is a separable function which correspond to the summation of local objective function F_i over all the nodes of the network. The $\mathbf{A}$ matrix is a block diagonal matrix that can be built as $\mathbf{A} = \text{diag}(\mathbf{A}_i, i \in \mathcal{N})$.

To find the solution of Problem (57) we look for the stationary points of the augmented Lagrangian function, which is defined as

$$L(\hat{\mathbf{x}}, \mathbf{z}, \boldsymbol{\lambda}; \mathbf{c}) = \sum_{i \in \mathcal{N}} \left(F_i(\hat{\mathbf{x}}_i) + \boldsymbol{\lambda}_i^T (\mathbf{A}_i \hat{\mathbf{x}}_i - \mathbf{z}_i) + \frac{1}{2} c_i \|\mathbf{A}_i \hat{\mathbf{x}}_i - \mathbf{z}_i\|^2 \right) \quad (58)$$

where the vector $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_i]_{i \in \mathcal{N}}$ collects the Lagrangian multipliers of all the nodes and, at the same way, the vector $\mathbf{c} = [c_i]_{i \in \mathcal{N}}$ collects the penalty parameters of all the nodes. These penalty parameters must always be positive and they weight the penalty term.

The stationary points of the augmented Lagrangian function (58) correspond to the Karush Kuhn Tucker (KKT) stationary points of Problem (57). Furthermore, if the object function is differentiable, these points also correspond to local minima [17].

The KKT stationary points are identified by conditions

$$\begin{aligned} \mathbf{A}\hat{\mathbf{x}} - \mathbf{z} &= \mathbf{0} \\ \mathbf{E}_i \left(\nabla F_i(\hat{\mathbf{x}}) + \mathbf{A}_i^T \boldsymbol{\lambda} \right) &= \mathbf{0}, \forall i \in \mathcal{N} \\ \boldsymbol{\lambda} &\perp \mathcal{Z} \\ \mathbf{z} &\in \mathcal{Z} \\ \hat{\mathbf{x}}_{k,k} &= \mathbf{a}_k, \forall k \in \mathcal{A} \end{aligned} \quad (59)$$

where we introduced the function

$$\mathbf{E}_i = \begin{cases} \mathbf{I}_{n(N_i+1)} & i \notin \mathcal{A} \\ \text{diag}(0, \mathbf{I}_{N_i}) \otimes \mathbf{I}_n & i \in \mathcal{A} \end{cases} \quad (60)$$

that it is used to neglect the equality in the first coordinate entry (of length n) when dealing with anchor nodes.

4.2.1 *ADMM iterative algorithm*

To look for stationary points of the augmented Lagrangian function, L, we perform an alternating search, which, at every iteration t, performs an update of vectors $\hat{\mathbf{x}}$, $\mathbf{z}$ and $\boldsymbol{\lambda}$. We define the values of these vectors at each iteration t by using the notation $\hat{\mathbf{x}}^t$, $\mathbf{z}^t$ and $\boldsymbol{\lambda}^t$.

First, starting points are selected for all of these variables. In particular, we set random initial points $\hat{\mathbf{x}}^0$, that need to verify the equiva-

lence $\hat{\mathbf{x}}^0_{j,i} = \hat{\mathbf{x}}^0_{i,i}$, $j \in \mathcal{N}_i$ and consequently $\boldsymbol{\lambda}^0 = \mathbf{0}$ and $\mathbf{z}^0 = \mathbf{A}\hat{\mathbf{x}}^0$.
At each iteration, the update follows as

$$\begin{aligned}
\hat{\mathbf{x}}^{t+1} &\in \arg\min_{\hat{\mathbf{x}} \in \mathcal{X}} L\left(\hat{\mathbf{x}}, \mathbf{z}^t, \boldsymbol{\lambda}^t; \mathbf{c}^t\right) \\
\mathbf{z}^{t+1} &\in \arg\min_{\mathbf{z} \in \mathcal{Z}} L\left(\hat{\mathbf{x}}^{t+1}, \mathbf{z}, \boldsymbol{\lambda}^t; \mathbf{c}^t\right) \\
\boldsymbol{\lambda}_i^{t+1} &= \boldsymbol{\lambda}_i^t + c_i^t\left(\mathbf{A}_i\hat{\mathbf{x}}_i^{t+1} - \mathbf{z}_i^{t+1}\right), \forall i \in \mathcal{N}
\end{aligned} \tag{61}$$

A special consideration must be done on the importance of the penalty terms c_i^t, that it is proven need to be well set in order to reach the convergence [2].

We will use the ADMM algorithm both with convex and non-convex functions. In the first case it is guaranteed that the limit point of (61) identifies a global minimum. In the non-convex scenario, the algorithm may find a local, rather than a global, minimum and this can affect the optimality of the localization solution.

4.2.2 *Distributed ADMM solution*

The iterative approach shown in Section 4.2.1 cannot be used in a distributed way because, as can be seen in equations (61), this approach needs a global knowledge of the whole variables set. We need to enable each node to update its variables, in particular the triplet $(\hat{\mathbf{x}}, \mathbf{z}, \boldsymbol{\lambda})$, in an autonomous way using information gathered from its neighbors.

Let's start looking at equations (61). In particular, we see that the third equation does not need to be reviewed since it can already be locally computed. We need, on the contrary, to revise the first and the second equation.

The update of the nodes positions, shown in the first equation of (61), can be formulated as the local update of variables $\hat{\mathbf{x}}_i^t$, $i \in \mathcal{N}$, according to

$$\hat{\mathbf{x}}_i^{t+1} \in \arg\min_{\hat{\mathbf{x}}_i \in \mathcal{X}_i} F_i\left(\hat{\mathbf{x}}_i\right) + \frac{1}{2} c_i^t \left(\hat{\mathbf{x}}_i - \mathbf{y}_i^t\right)^T \mathbf{D}_i \left(\hat{\mathbf{x}}_i - \mathbf{y}_i^t\right) \tag{62}$$

where $\mathbf{x}_i$ belongs to the set

$$\mathcal{X}_i = \begin{cases} \mathbb{R}^{n(N_i+1)} & i \notin \mathcal{A} \\ \left\{\hat{\mathbf{x}}_i \in \mathbb{R}^{n(N_i+1)} | \hat{\mathbf{x}}_{i,i} = \mathbf{a}_i\right\} & i \in \mathcal{A} \end{cases} \tag{63}$$

and the $\mathbf{y}_i^t$ variables are defined as

$$\mathbf{y}_i^t = \mathbf{D}_i^{-1}\mathbf{A}_i^T\left(\mathbf{z}_i^t - \tilde{\boldsymbol{\lambda}}_i^t\right), \qquad \tilde{\boldsymbol{\lambda}} = \frac{\boldsymbol{\lambda}_i^t}{c_i^t} \tag{64}$$

Both in the $\hat{\mathbf{x}}_i^{t+1}$ and $\mathbf{y}_i^t$ definitions we used the $\mathbf{D}_i$ matrix, which is

$$\mathbf{D}_i = \mathbf{A}_i^T\mathbf{A}_i = \begin{bmatrix} 2N_i & \mathbf{0}_{N_i}^T \\ \mathbf{0}_{N_i} & 2\mathbf{I}_{N_i} \end{bmatrix} \otimes \mathbf{I}_n \tag{65}$$

This matrix is by construction positive definite, with eigenvalues 2 and $2N_i$. The role of the quadratic term in (62) is that of pushing the solution towards $\mathbf{y}_i^t$, defined in (64).

We can now continue with the second line of (61), which performs the update of auxiliary variables $\mathbf{z}$. To perform this update in a distributed fashion, we use the following equation

$$\mathbf{z}^{t+1} = \mathbf{L}_{\mathcal{Z}} \left(\mathbf{A}\hat{\mathbf{x}}^{t+1} + \tilde{\boldsymbol{\lambda}}^t \right) \tag{66}$$

where we denote the projection associated with the linear space $\mathcal{Z}$ using the function $\mathbf{L}_{\mathcal{Z}}$.

Messages shared by nodes are indicated as $\mathbf{m}_i$. In particular, they are defined as

$$\mathbf{m}_i^{t+1} = \begin{bmatrix} \left[\mathbf{m}_{1,i,j}^t\right]_{j\in\mathcal{N}_i} \\ \left[\mathbf{m}_{2,i,j}^t\right]_{j\in\mathcal{N}_i} \end{bmatrix} = \mathbf{A}_i \mathbf{x}_i^{t+1} + \tilde{\boldsymbol{\lambda}}_i^t \tag{67}$$

This definition enables us to rewrite (66) in a form that allows the local computation and, consequently, the application to a distributed scenario.

$$\begin{aligned} \mathbf{z}_{1,i,j}^{t+1} &= \frac{1}{2}\left(\mathbf{m}_{1,i,j}^{t+1} - \mathbf{m}_{1,j,i}^{t+1}\right) \\ \mathbf{z}_{2,i,j}^{t+1} &= \frac{1}{2}\left(\mathbf{m}_{2,i,j}^{t+1} + \mathbf{m}_{2,j,i}^{t+1}\right) \end{aligned} \tag{68}$$

This concludes the derivation of locally update functions that enables the development of a distributed ADMM algorithm, which is resumed in Algorithmus 2.

for $t = 0$ to ∞ **do**
  if $t = 0$ **then**
    Initialize local position $\hat{\mathbf{x}}_i^0$
    Build messages $\mathbf{m}_i^0 = \mathbf{A}_i \hat{\mathbf{x}}_i^0$
  else
    Update local positions $\hat{\mathbf{x}}_i^t$ via (62)
    Build messages $\mathbf{m}_i^t = \mathbf{A}_i \hat{\mathbf{x}}_i^t + \boldsymbol{\lambda}_i^{t-1} / c_i^{t-1}$
  end
  $\Rightarrow$ Broadcast values $\mathbf{m}_{\cdot,i,j}^t$ to region $j \in \mathcal{N}_i$
  $\Leftarrow$ Receive values $\mathbf{m}_{\cdot,j,i}^t$ from region $j \in \mathcal{N}_i$
  Evaluate the mixing values $\mathbf{z}_i^t$ via (68)
  if $t = 0$ **then**
    Reset memory $\boldsymbol{\lambda}_i^0 = \mathbf{0}$
  else
    Update memory $\boldsymbol{\lambda}_i^t = \boldsymbol{\lambda}_i^{t-1} + c_i^{t-1} \left(\mathbf{A}_i \hat{\mathbf{x}}_i^t - \mathbf{z}_i^t \right)$
  end
  Eventually update c_i^t
  Evaluate $\mathbf{y}_i^t = \mathbf{D}_i^{-1} \mathbf{A}_i^T \left(\mathbf{z}_i^t - \boldsymbol{\lambda}_i^t / c_i^t \right)$
end

Algorithmus 2 : Parallel processing algorithm at node i. Variable t represents the iteration number.

5

APPLICATION OF ADMM IN A CONVEX SCENARIO

In Chapter 3 we saw how it is possible to relax the optimization problem using a convex relaxation of the original localization problem function. Now, we want to apply the ADMM algorithm in this scenario. We must pay attention, in particular, to two aspects. The first regards the update function of the position vector $\mathbf{x}_i$, that, as shown in (62), needs the resolution of a minimization problem. To this aim we use the *fmincon* solver provided by the *MATLAB Optimization Toolbox*.

To reduce the solver computation time and increase the solution accuracy, we will need to provide to the solver the gradient and hessian of the minimization problem objective function, that we will compute in the next section.

The second aspect concerns the penalty parameters c_i that, in this scenario, will be fixed values. The convex relaxation guarantees, in fact, that the ADMM algorithm will converge to the global minimum.

To simplify the application of the ADMM algorithm, we will work with a problem which is very similar to the convex problem (21).

$$\begin{aligned} &\min_{\mathbf{x}} \sum_{i\sim j} \frac{1}{2} d^2_{\bar{C}_{ij}} \left(\mathbf{x}_i - \mathbf{x}_j\right) \\ &\text{s.t. } \mathbf{x}_k = \mathbf{a}_k, \forall k \in \mathcal{A} \end{aligned} \tag{69}$$

where $\mathbf{a}_k$ is the n-dimensional position of the k-th anchor node.

Initially, we need to derive the local objective function used in the *Distributed ADMM algorithm*. This function can be derived from the ADMM objective function, defined in Problem (57) as $F(\hat{\mathbf{x}})$, considering (54)

$$F(\hat{\mathbf{x}}) = \sum_{i\in\mathcal{N}} F_i(\hat{\mathbf{x}}_i) = \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}_i} f_{i,j}\left(\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|\right) \tag{70}$$

It is possible to rewrite the objective function of the convex problem as

$$\sum_{i\sim j} \frac{1}{2} d^2_{\bar{C}_{i,j}} \left(\mathbf{x}_i - \mathbf{x}_j\right) = \sum_{i\in\mathcal{N}} \left(\sum_{j\in\mathcal{N}_i} \frac{1}{4} d^2_{\bar{C}_{i,j}} \left(\mathbf{x}_i - \mathbf{x}_j\right) \right) \tag{71}$$

where the $\frac{1}{4}$ term is used because the two summations consider two times each edge. From this equality and from (70) we can see that

$$f_{i,j}\left(\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|\right) = \frac{1}{4} d^2_{\bar{C}_{i,j}} \left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right) \tag{72}$$

which corresponds to the *building block* of the problem objective function. This allows us to define the local problem objective function as

$$F_i(\hat{\mathbf{x}}_i) = \sum_{j \in \mathcal{N}_i} \frac{1}{4} d^2_{\mathcal{C}_{i,j}}(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}) \tag{73}$$

Given this, we can write the node position update function, which we just defined in (62), as

$$\hat{\mathbf{x}}_i^{t+1} \in \underset{\hat{\mathbf{x}}_i \in \mathcal{X}_i}{\arg\min} \left(\sum_{j \in \mathcal{N}_i} \frac{1}{4} d^2_{\mathcal{C}_{i,j}}(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}) \right) + \frac{1}{2} c_i^t (\hat{\mathbf{x}}_i - \mathbf{y}_i^t)^\top \mathbf{D}_i (\hat{\mathbf{x}}_i - \mathbf{y}_i^t) \tag{74}$$

This function will be used in the following sections to derive the gradient and hessian functions that are useful to improve the computation time of the *solver* used by the ADMM algorithm.

5.1 GRADIENT OF THE NODE POSITION UPDATE FUNCTION

To simplify the derivation of (74) we start deriving the first term, which corresponds to the F_i function. Considering the definition of $d_{\mathcal{C}_{i,j}}$ given in (15) and the following analytical solution given in (23), we can rewrite function F_i as

$$F_i(\hat{\mathbf{x}}_i) = \sum_{j \in \mathcal{N}_i} \frac{1}{4} \left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\| - r_{ij} \right)^2 1\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\| - r_{ij} \right) \tag{75}$$

Since the $\hat{\mathbf{x}}_{i,i}$ term appears in every term of the summation, the derivative of F_i with respect to $\hat{\mathbf{x}}_{i,i}$ will be defined by a summation, which is

$$\frac{\partial}{\partial \hat{\mathbf{x}}_{i,i}} F_i(\hat{\mathbf{x}}_i) = \sum_{j \in \mathcal{N}_i} \frac{1}{2} \left(1 - \frac{r_{ij}}{\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\|} \right) (\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}) \, 1\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\| - r_{ij} \right) \tag{76}$$

With this equation we can calculate the first two elements of the gradient vector. For the others, which correspond to the derivative of F_i with respect to the x and y components of the j-th neighbor location, we calculate

$$\frac{\partial}{\partial \hat{\mathbf{x}}_{i,j}} F_i(\hat{\mathbf{x}}_i) = -\frac{1}{2} \left(1 - \frac{r_{ij}}{\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\|} \right) (\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}) \, 1\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\| - r_{ij} \right) \tag{77}$$

We now look at the second term of the objective function. Considering that the $\mathbf{D}_i$ matrix is symmetric, the gradient of the cost function can be computed as

$$\frac{\partial}{\partial \hat{\mathbf{x}}_i} f_c(\hat{\mathbf{x}}_i) = c_i^t \cdot \mathbf{D}_i (\hat{\mathbf{x}}_i - \mathbf{y}_i^t) \tag{78}$$

So the gradient of the node position update function, f_o, is

$$\nabla f_o(\hat{\mathbf{x}}_i) = \begin{bmatrix} \frac{\partial}{\partial \hat{\mathbf{x}}_{i,i}} F_i(\hat{\mathbf{x}}_i) \\ \left[\frac{\partial}{\partial \hat{\mathbf{x}}_{i,j}} F_i(\hat{\mathbf{x}}_i)\right]_{j \in \mathcal{N}_i} \end{bmatrix} + \frac{\partial}{\partial \hat{\mathbf{x}}_i} f_c(\hat{\mathbf{x}}_i) \tag{79}$$

5.2 HESSIAN OF THE NODE POSITION UPDATE FUNCTION

After calculating the gradient of the node position update function, we need to provide to the solver the Hessian function. Like in the previous section we initially consider the first term of (74), which corresponds to (75). By deriving (76) and (77), we find

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,i}^2} F_i(\hat{\mathbf{x}}_i) = \frac{1}{2} \sum_{j \in \mathcal{N}_i} \left[r_{ij} \frac{\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)^T}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|^3} + \left(1 - \frac{r_{ij}}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|}\right) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] s_{i,j} \tag{80}$$

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,j} \hat{\mathbf{x}}_{i,i}} F_i(\hat{\mathbf{x}}_i) = \frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,i} \hat{\mathbf{x}}_{i,j}} F_i(\hat{\mathbf{x}}_i) = -\frac{1}{2} \left[r_{ij} \frac{\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)^T}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|^3} + \left(1 - \frac{r_{ij}}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|}\right) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] s_{i,j} \tag{81}$$

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,j}^2} F_i(\hat{\mathbf{x}}_i) = \frac{1}{2} \left[r_{ij} \frac{\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)\left(\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right)^T}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|^3} + \left(1 - \frac{r_{ij}}{\left|\left|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\right|\right|}\right) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] s_{i,j} \tag{82}$$

where $s_{i,j} = 1\left(\|\hat{\mathbf{x}}_{i,i} - \hat{\mathbf{x}}_{i,j}\| - r_{ij}\right)$. The hessian of the second term can easily be computed as

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_i^2} f_c(\hat{\mathbf{x}}_i) = c_i^t \cdot \mathbf{D}_i \tag{83}$$

Collecting all the computed parts, we can finally define the hessian of the node position update function as

$$H_f(\hat{\mathbf{x}}_i) = \begin{bmatrix} \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,i}^2} & \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,i} \hat{\mathbf{x}}_{i,1}} & \cdots & \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,i} \hat{\mathbf{x}}_{i,N_i}} \\ \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,1} \hat{\mathbf{x}}_{i,i}} & \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,1} \hat{\mathbf{x}}_{i,1}} & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,N_i} \hat{\mathbf{x}}_{i,i}} & 0 & 0 & \frac{\partial^2 F_i}{\partial \hat{\mathbf{x}}_{i,N_i} \hat{\mathbf{x}}_{i,N_i}} \end{bmatrix} + c_i^t \cdot \mathbf{D}_i \tag{84}$$

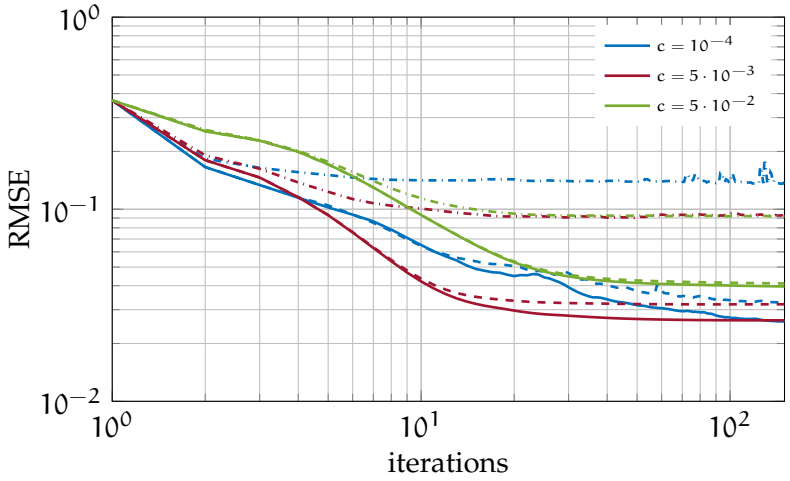


Figure 12: Average RMSE performances of ADMM algorithm in convex scenario applied to the 40 nodes network. Different noise realizations are considered. Continuous lines indicate performances for noise standard deviation $\sigma = 10^{-3}$, dashed lines for $\sigma = 10^{-2}$, dash-dotted lines for $\sigma = 10^{-1}$

5.3 NUMERICAL SIMULATIONS

The ADMM algorithm considered in this section is structured like Algorithmus 2 and no updates are performed on the penalty parameters $\{\mathbf{c}_i\}_{i\in\mathcal{N}}$. The only degree of freedom we have is the value of these parameters, that impacts on the RMSE performances.

In this section we will also identify the best parameters choices, that will be used in Chapter 8 to compare all the algorithms introduced in this thesis work. Like in Chapter 3, we will perform our tests on three different networks.

5.3.1 *Simulations on the 40 nodes network*

We want to test the algorithm considering 50 different noise realizations for each noise standard deviation, $\sigma = 10^{-3}$, $\sigma = 10^{-2}$ and $\sigma = 10^{-1}$.

It is fundamental to understand how the choice of the penalty parameters impacts on the RMSE performances. In Figure 12 we show the RMSE performance for three c choices, where $\{\mathbf{c}_i\}_{i\in\mathcal{N}} = c$.

The first choice, $c = 10^{-4}$, leads to bad results: the RMSE, in the $\sigma = 10^{-1}$ case, converges to an high value while in the other cases it converges to good values but too slowly.

The $c = 5 \cdot 10^{-2}$ choice, leads to a good RMSE value at convergence in the $\sigma = 10^{-1}$ case but in the other two cases, the RMSE value at convergence is bad and, for all the noise levels, the convergence is slow.

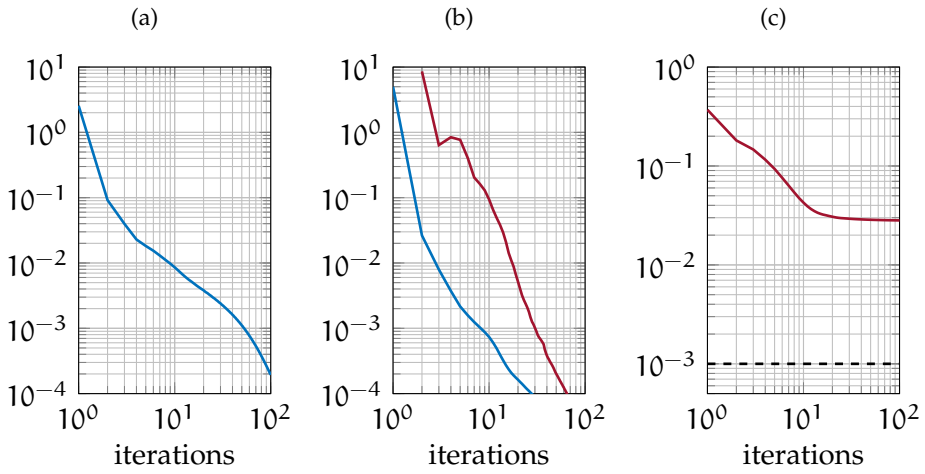


Figure 13: 40 nodes network. (a) target function value $f(\hat{x})$ over the iterations number; (b) primal feasibility gap $P(t)$ (red line) and stationary gap $S(t)$ (blue line); (c) RMSE performance

Finally, the best choice is $c = 5 \cdot 10^{-3}$, which guarantees a good RMSE value for all the noise level cases, and a fast convergence, in only 20 iterations.

It is interesting to examine other aspects of the algorithm convergence and in order to do this, we consider a single noise realization of the 40 nodes network, with $\sigma = 10^{-3}$. The penalty parameter is set to the optimal value $c = 5 \cdot 10^{-3}$.

In Figure 13a, we can observe the evolution of the target function $f(\mathbf{x}) = \sum_{i \in \mathcal{N}} f_{i,j}\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)$ minimization, as a function of the iterations number.

The convergence is also illustrated in Figure 13b, showing the fast convergence to zero of the primal feasibility and stationary gaps, according to the first two lines of (59).

$$\begin{aligned} P(t) &= \left|\left|\mathbf{A}\mathbf{x}^t - \mathbf{z}^t\right|\right|^2 \\ S(t) &= \sum_{i \in \mathcal{N}} \left|\left|\mathbf{E}_i\left(\nabla F_i\left(\mathbf{x}_i^t\right) + \mathbf{A}_i^T(\lambda_i^t)\right|\right|^2 \end{aligned} \tag{85}$$

The convergence to zero of these two functions, in the convex scenario, is a guarantee that the global minimum is reached. in fact, the others condition in (59) are valid throughout the process.

Finally, from Figure 13c we can see that the RMSE reaches its minimum value after 20 iterations.

5.3.2 *Simulations on large networks*

Now we want to find a good penalty parameter value for our large networks, which are composed of 500 nodes in the first case and 1000 nodes in the second one.

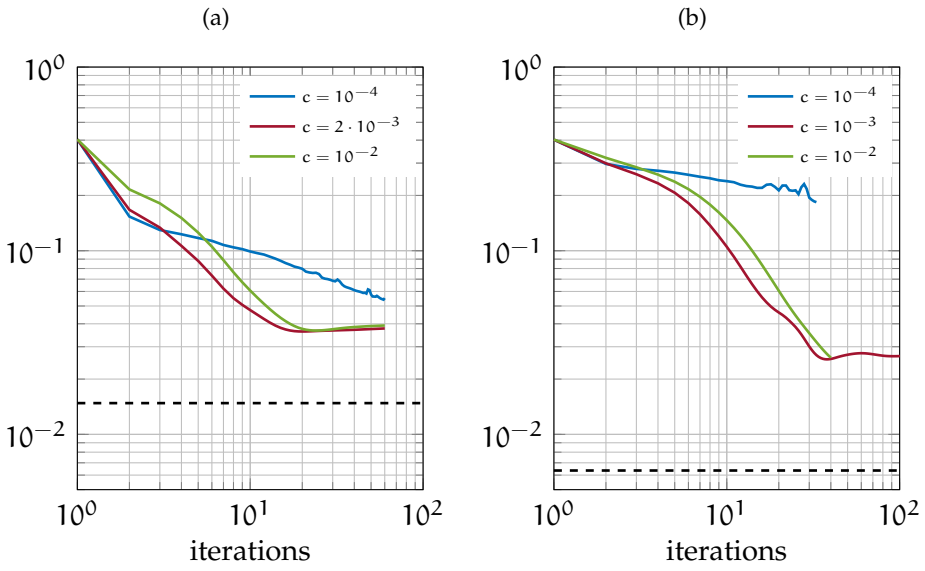


Figure 14: RMSE performances of ADMM algorithm in convex scenario, for different penalty term values. Dashed lines indicate CRLB. (a) 500 nodes network; (b) 1000 nodes network.

In Figure 14a we can see the RMSE performances in the 500 nodes network, for three different penalty parameter values. For $c = 10^{-4}$ we have bad performance since the algorithm convergence is very slow compared with the other two choices. With $c = 2 \cdot 10^{-3}$ and $c = 10^{-2}$, the algorithm converges to almost the same RMSE value but the best choice for the c parameter is the first, that guarantees a faster convergence, in less than 20 iterations.

The same test is finally performed on the 1000 nodes network, which is the larger in which we perform our simulations. In this case, RMSE performances are shown in Figure 14b.

Looking at the RMSE performance we can identify the best choice between the proposed penalty parameters, which is $c = 10^{-3}$. In this case the algorithm converges after 50 iterations.

6

APPLICATION OF ADMM IN A NON-CONVEX SCENARIO

In this chapter we apply the ADMM algorithm discussed in Chapter 4 in a non-convex scenario, like the one proposed by Erseghe [11].

In this case the convergence of the algorithm will be more difficult because the functions we need to optimize may present several local minima and the algorithm may get stuck on one of them. For this reason we need to define a good method to update the penalty parameters $\{c_i\}_{i\in\mathcal{N}}$ in order to guarantee the convergence. We also need to put a limit to the grown of the Lagrange multiplier.

The first thing we need to define, is the non-convex problem, which is

$$\begin{aligned} &\min_{\mathbf{x}} \sum_{i\sim j} f_{i,j}\left(\left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|\right) \\ &\text{s.t. } \mathbf{x}_k = \mathbf{a}_k,\ \forall k \in \mathcal{A} \end{aligned} \tag{86}$$

where we use the following log-likelihood function

$$f_{i,j}\left(\left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|\right) = \frac{1}{\sigma_{i,j}^2}\left(r_{ij} - \left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|\right)^2 \tag{87}$$

where $\sigma_{i,j}^2$ is the noise standard deviation that we assume always equal to 1 without loss in generality. The local problem objective function is easily defined as

$$F_i\left(\mathbf{x}_i\right) = \sum_{j\in\mathcal{N}_i} f_{i,j}\left(\left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|\right) = \sum_{j\in\mathcal{N}_i}\left(r_{ij} - \left|\left|\mathbf{x}_i - \mathbf{x}_j\right|\right|\right)^2 \tag{88}$$

From this equation we can see that we are dealing with a non-convex quadratic problem. Function (88) is used by the algorithm to update the positions estimates of each node, like in the convex scenario.

Like in the previous chapter we need to calculate the gradient and hessian functions that will be used by the solver to improve accuracy and computation time.

6.1 GRADIENT OF THE NODE POSITION UPDATE FUNCTION

We need to compute the gradient of the local objective function F_i with respect to the position of the generic node i, which is indicated as $\hat{\mathbf{x}}_{i,i}$, and with respect to the position of the j-th neighbor of i, $\hat{\mathbf{x}}_{i,j}$.

$$\begin{aligned}\frac{\partial}{\partial \hat{\mathbf{x}}_{i,i}} F_i(\hat{\mathbf{x}}_i) &= -\sum_{j\in\mathcal{N}_i} 2\left(\frac{r_{ij}}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||}-1\right)(\hat{\mathbf{x}}_{i,j}-\hat{\mathbf{x}}_{i,j})\\ \frac{\partial}{\partial \hat{\mathbf{x}}_{i,j}} F_i(\hat{\mathbf{x}}_i) &= 2\left(\frac{r_{ij}}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||}-1\right)(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})\end{aligned} \tag{89}$$

Since the gradient of the cost is the same that we calculated in (78), we can compute the gradient of the node position update function like in (79), but considering the derivatives of F_i calculated in (89).

6.2 HESSIAN OF THE NODE POSITION UPDATE FUNCTION

Like in the previous chapter we proceed with the calculation of the hessian function. We initially compute the second derivatives of F_i.

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,i}^2} F_i(\hat{\mathbf{x}}_i) = 2\sum_{j\in\mathcal{N}_i}\left[r_{ij}\frac{(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})^T}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||^3}+ \right.$$
$$\left. +\left(1-\frac{r_{ij}}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||}\right)\begin{pmatrix}1&0\\0&1\end{pmatrix}\right] \tag{90}$$

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,j}\hat{\mathbf{x}}_{i,i}} F_i(\hat{\mathbf{x}}_i) = \frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,i}\hat{\mathbf{x}}_{i,j}} F_i(\hat{\mathbf{x}}_i) = -2\left[r_{ij}\frac{(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})^T}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||^3}+ \right.$$
$$\left. +\left(1-\frac{r_{ij}}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||}\right)\begin{pmatrix}1&0\\0&1\end{pmatrix}\right] \tag{91}$$

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_{i,j}^2} F_i(\hat{\mathbf{x}}_i) = 2\left[r_{ij}\frac{(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})(\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j})^T}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||^3}+ \right.$$
$$\left. +\left(1-\frac{r_{ij}}{||\hat{\mathbf{x}}_{i,i}-\hat{\mathbf{x}}_{i,j}||}\right)\begin{pmatrix}1&0\\0&1\end{pmatrix}\right] \tag{92}$$

Then we can calculate the hessian by using equation (84) with these equations.

6.3 UPDATE OF THE PENALTY PARAMETERS

In this scenario we are dealing with non-convex functions, as we described in the first part of the chapter. This means that in some cases

the algorithm suggested in Chapter 4 may fail to converge, in particular when a feasible but not optimal point is approached. An efficient countermeasure to this problem is the update of the penalty terms c_i.

A good way to solve the non-convergence problem is, in fact, to increase the value of these parameters while the algorithm proceeds on its iterations, as suggested by several works [2] [1] [4] [11].

In order to introduce the penalty parameters update function, we define the current local primal gap as

$$\Gamma_i^t = \left|\left|\mathbf{A}_i\mathbf{x}_i^t - \mathbf{z}_i^t\right|\right|_\infty \tag{93}$$

where $\|\cdot\|_\infty$ is an infinity norm, that coincides with the maximum absolute value.

A good idea is to decide to update the local penalty values if the local primal gap, Γ_i do not decrease sufficiently. We need then, to define a threshold that establishes when the decrease is sufficient, through the parameter θ_c. We also need to guarantee that the penalty parameters increase through the network if at least one local primal gap is not sufficiently decreasing.

We collect all of these requirements in the following penalty parameters update function

$$c_i^t = \left(\max_{j\in\mathcal{N}_i\cup\{i\}} c_j^{t-1}\right)\cdot\begin{cases} 1 & ,\Gamma_i^t \leqslant \theta_c\Gamma_i^{t-1} \\ \delta_c & ,\text{otherwise}\end{cases} \tag{94}$$

where $\delta_c > 1$ but close to 1 and where $\theta_c < 1$.

This update function requires the exchange of the penalty parameters c_i between nodes, since it needs to maximize c_j^{t-1}, $j \in \mathcal{N}_i \cup \{i\}$. This data exchange can be done at the same time with the exchange of values $\mathbf{m}^t$. Furthermore, equation (94) guarantees that for a properly descending primal local gap Γ_i^t, the penalty constant c_i through the network will converge to a unique value.

6.4 CONVERGENCE OF THE ADMM ALGORITHM IN NON-CONVEX SCENARIOS

We described the peculiarities of non-convex scenarios in the previous section. The update of the penalty parameters introduced in equation (94) is however not sufficient to guarantee the convergence of the algorithm. In this section we complete the discussion on a modified ADMM algorithm that guarantees the converge also in non-convex scenarios.

In addition to the penalty parameters update function, we need to bound the Lagrange multipliers and the search for updates $\mathbf{x}_i^t$, in order to guarantee the convergence.

To bound the Lagrange multipliers, we substitute the update function

$$\boldsymbol{\lambda}_i^t = \boldsymbol{\lambda}_i^{t-1} + c_i^{t-1}\left(\mathbf{A}_i\hat{\mathbf{x}}_i^t - \mathbf{z}_i^t\right) \tag{95}$$

with

$$\boldsymbol{\lambda}_i^t = \mathcal{P}_{\lambda_{max}}\left[\boldsymbol{\lambda}_i^{t-1} + c_i^{t-1}\left(\mathbf{A}_i\hat{\mathbf{x}}_i^t - \mathbf{z}_i^t\right)\right] \tag{96}$$

where the function $\mathcal{P}_{\lambda_{max}}$ performs a clipping of the vector entries in the range $[-\lambda_{max}, \lambda_{max}]$. To bound the search for positions updates $\mathbf{x}_i^t$ we replace the set

$$\mathcal{X}_i = \begin{cases} \mathbb{R}^{n(N_i+1)} & i \notin \mathcal{A} \\ \left\{\hat{\mathbf{x}}_i \in \mathbb{R}^{n(N_i+1)} | \hat{\mathbf{x}}_{i,i} = \mathbf{a}_i\right\} & i \in \mathcal{A} \end{cases} \tag{97}$$

with

$$\mathcal{X}_i = \begin{cases} \{\hat{\mathbf{x}}_i | F_i(\hat{\mathbf{x}}_i) \leqslant F(\tilde{\mathbf{p}})\} & i \notin \mathcal{A} \\ \{\hat{\mathbf{x}}_i | F_i(\mathbf{x}_i) \leqslant F(\tilde{\mathbf{p}}), \hat{\mathbf{x}}_{i,i} = \mathbf{a}_i\} & i \in \mathcal{A} \end{cases} \tag{98}$$

where $\tilde{\mathbf{p}}$ is some initial position estimate for which $F(\tilde{\mathbf{p}}) < \infty$.

It is proved [11] that for a good choice of parameters, the algorithm converges. In particular they must be chosen in a correct range, namely $c_i^0 > 0$, $\delta_c > 1$, $0 < \theta_c < 1$ and $\lambda_{max} > 0$.

It is also proved that the algorithm converges to a local minimum if there is a finite iteration value t_0 such that for $t \geqslant t_0$ the penalty parameters $\{c_i^t\}_{i\in\mathcal{N}}$ are not updated and the Lagrange multipliers are not clipped.

We must take care of the choice of the initial value of $c_{i,i\in\mathcal{N}}$ since if this value is too high, the convergence will be slow. A good way of approaching the problem, is to set the penalty constants to a small value and then slowly increase them, setting $\delta_c \simeq 1$. However, to prevent unwanted clipping actions on the Lagrange multipliers, we set δ_{max} to a big value, typically 10^3 or 10^4, but obviously in dependence with the considered case.

6.5 NUMERICAL SIMULATION

We now perform numerical simulations on all the previous seen networks, in order to find good parameters for the ADMM algorithm in non-convex scenario. The algorithm considered in this section, corresponds to Algorithmus 2, with local problem objective functions defined as (88), Lagrange multipliers update function (96) and penalty parameters update function (94).

All the coming simulations have been computed with fixed values of $\delta_c = 1.01$, $\theta_c = 0.98$, $\lambda_{max} = 10^4$. The only freedom degree is so represented by the starting penalty parameter $\{c_i^0\}_{i\in\mathcal{N}} = c$.

6.5.1 *Simulations on the 40 nodes network*

We want to see how the algorithm behaves when considering different noise standard deviations. To do this we consider three different

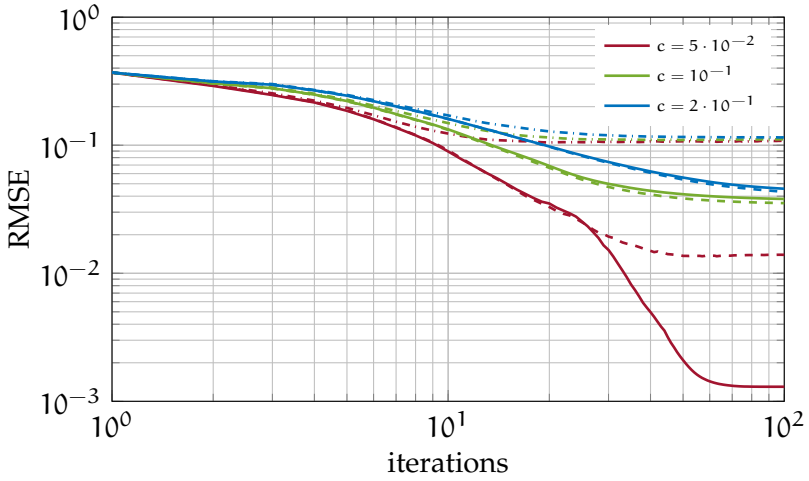


Figure 15: Average RMSE performances of ADMM algorithm in non-convex scenario applied to the 40 nodes network. Different noise realizations are considered. Continuous lines indicate performances for noise standard deviation $\sigma = 10^{-3}$, dashed lines for $\sigma = 10^{-2}$, dash-dotted lines for $\sigma = 10^{-1}$

noise levels ($\sigma = 10^{-1}$, $\sigma = 10^{-2}$, $\sigma = 10^{-3}$) and 50 noise realizations for each of these levels.

From Figure 15 we can observe the RMSE performance for three different starting penalty parameter choices.

All the choices guarantee a good RMSE convergence value in the $\sigma = 10^{-1}$ case. Looking at the two other noise levels, $\sigma = 10^{-2}$ and $\sigma = 10^{-3}$, we can observe that the $c = 5 \cdot 10^{-2}$ choice guarantees better performances in both cases.

Note that the algorithm converges faster when the noise level is high, but the achieved RMSE value is high. With less noise, the convergence is slower but we achieve better RMSE values.

Now, we consider a single noise realization of the 40 nodes network. The noise standard deviation, in this case, is $\sigma = 10^{-2}$ and the starting penalty parameter is set to the optimum value that we just found, $c_i^0 = 5 \cdot 10^{-2}, \forall i \in \mathcal{N}$.

In Figure 16a we can see how the target function, defined as $f(\mathbf{x}) = \sum_{i \in \mathcal{N}} f_{i,j}\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\right)$, decreases as the number of iterations grows.

The RMSE performance is shown in Figure 16b, where we observe that the the convergence value is very close to the CRLB (dashed line). The algorithm converges after 40 iterations.

Finally, in Figure 16c we see the increase of the penalty parameter value over the network, as the iteration number grows.

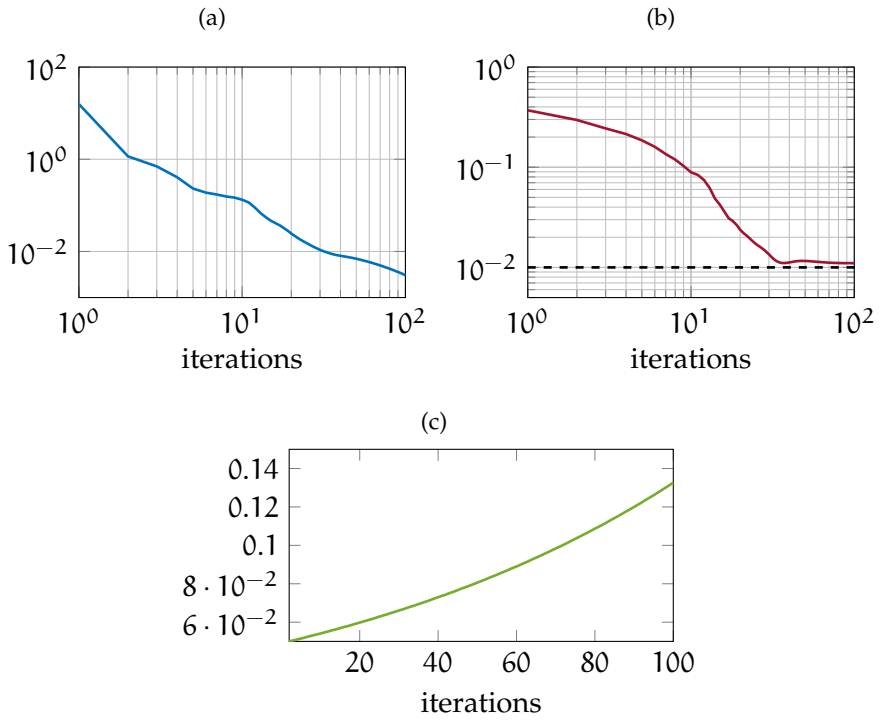


Figure 16: 40 nodes network. (a) target function value $f(\hat{\mathbf{x}})$ over the iterations number; (b) RMSE performance; (c) update of penalty coefficients.

6.5.2 *Simulations on large networks*

Like in the previous chapters we want to examine the algorithm performances in large networks. We perform simulations of the ADMM algorithm in non-convex scenario on two large networks: the first has 500 nodes, of which 10 are anchors; the second has 1000 nodes, of which 20 are anchors.

The RMSE performance on the 500 nodes network is shown in Figure 17a for different penalty parameters starting value $\{c_i^0\} = c$, $\forall i \in \mathcal{N}$. We can observe that the best RMSE value is approached with $c = 5 \cdot 10^{-2}$, in less than 40 iterations.

In Figure 17b we can see the RMSE performance on the 1000 nodes network, instead. In this case the best starting value for the penalty parameters is $c = 2 \cdot 10^{-2}$ that allows the algorithm to converge to an RMSE value which is close to the bound provided by the CRLB, in less than 200 iterations.

We observe that the number of iterations required by the algorithm to converge increases with the network size and the noise level.

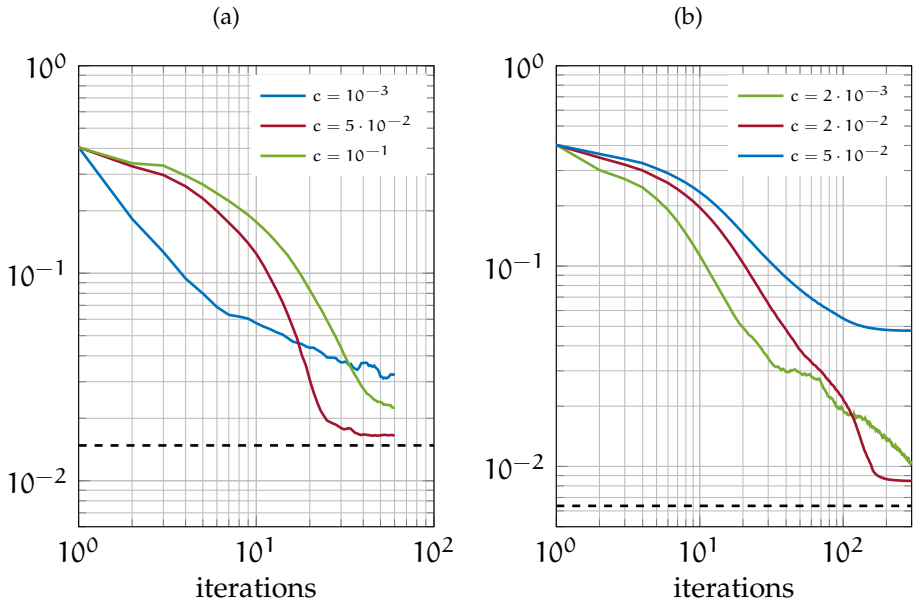


Figure 17: RMSE performances of ADMM algorithm in non-convex scenario, for different penalty term values. (a) 500 nodes network; (b) 1000 nodes network.

7 AN HYBRID SOLUTION

The proposal of this thesis work, introduced in the first chapter, is to analyze two localization approaches. We saw in Chapter 3 how we can relax the original localization problem in order to find a convex function to optimize. This solution presents some problems, as we have seen in the numerical simulation sections, because it does not allow to reach good RMSE performances.

In Chapter 5 we introduced a way to reduce the number of iterations required by the algorithm introduced in Chapter 3 to converge, using an ADMM approach.

In Chapter 6 we exploited the ADMM algorithm considering the original non convex localization problem.

To motivate the approach that we will introduce in this chapter, we now compare the performances found through numerical simulations in the previous chapters, considering the best found penalty parameters, c. More detailed comparisons will be discussed in Chapter 8, so we now compare the three presented solutions only in an example case.

The considered network is composed of 40 nodes, of which 10 are anchors nodes. The noise level is $\sigma = 10^{-3}$.

We refer to the algorithm introduced in Chapter 3 as *SF*[1]. The ADMM approach in convex scenario, which is an enhancement of the *SF* algorithm, is named *ADMM-SF*. We refer instead to the ADMM algorithm in non-convex scenario, discussed in Chapter 6, as *ADMM*.

From Figure 18 we can note that the ADMM-SF approach significantly improves the performance of the SF algorithm. The RMSE value at convergence is slightly better, in the considered network, but we can generally say that the two algorithms achieve the same RMSE value at convergence. However the number of iterations required by ADMM-SF to converge is much lower than the SF algorithm. SF converges after 100 iterations, while ADMM-SF requires only 15 iterations.

As we said, the RMSE value achieved by these two algorithms is not too good, as we can note looking to the CRLB bound, indicated in the figure by the dashed line.

A better result is provided by the ADMM algorithm in non-convex scenario which can reach an RMSE value which is close to the bound. The algorithm converges in a little more than 60 iterations but the RMSE value reached by this algorithm can motivate its usage.

1 due to the name of the work by Soares, Xavier, and Gomes [24]

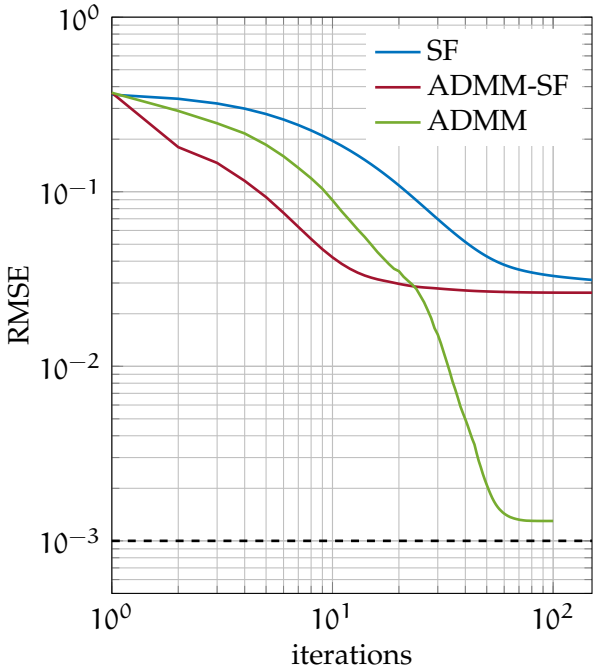


Figure 18: 40 nodes network with noise level $\sigma = 10^{-3}$. Comparison of the three introduced localization algorithms performances.

The basic idea is that we can exploit the fast convergence of the ADMM-SF algorithm to enhance the performance of the ADMM algorithm in the first part of its running. The proposed algorithm will start considering the convex relaxation of the original localization problem. When the algorithm begin to converge, it *switches* to the non-convex problem, in order to refine the achieved results and reach a better RMSE value.

7.1 DETECTION OF THE SWITCH MOMENT

In the hybrid ADMM algorithm we distinguish between two operational modes

- *convex mode*: In this mode the algorithm runs optimizing the relaxed convex problem in a distributed ADMM fashion;
- *non-convex mode*: In this mode the algorithm runs optimizing the original non-convex localization problem in a distributed ADMM fashion.

As we said, the algorithm starts optimizing the convex problem. The considered penalty parameter during this stage, is called c^{co} and, since the algorithm does not perform the update of this parameter in the convex scenario, this value is constant when the algorithm is running in *convex mode*. Defining the set

$$\mathcal{CM} = \{t : \text{iteration } t \text{ performed in convex mode}\} \tag{99}$$

whe can say that

$$c_i^t = c^{co}, \forall t \in \mathcal{CM} \tag{100}$$

for a generic node i.

We also define with c^{nc} the penalty parameter used by the hybrid algorithm when switching to the *non-convex mode*. In this case, the algorithm performs the update of this parameter like discussed in Section 6.3, so this penalty parameter will not remain constant during the execution in non-convex mode.

By defining with t_{sw} the time instant in which the algorithm switches from the convex mode to the non-convex mode, we can write

$$c_i^{t_{sw}} = c^{nc} \tag{101}$$

The nodes localization takes place in a distributed fashion and for this reason each node must detect autonomously when is the time to switch to the non-convex mode. It is important to find a good measure that results helpful to the node to detect when to switch to the non-convex mode. We define the current local primal gap as

$$\Gamma_i^t = \left|\left|\mathbf{A}_i \hat{\mathbf{x}}_i^t - \mathbf{z}_i^t\right|\right|_\infty \tag{102}$$

with $\|\cdot\|_\infty$ being an infinity norm, which corresponds to the maximum absolute value.

We can exploit this value to detect the switching moment. In order to do this, we define a threshold τ . Each node must switch to the non-convex mode when $\Gamma_i^t < \tau$.

The threshold value is an important parameter, indeed if its value is too high, nodes rapidly switch to the non-convex mode, and the benefits of this approach are null. However, if the threshold is too low, the nodes switch to the non-convex mode too slowly, and the algorithm requires lots of iterations to convergence.

The hybrid ADMM algorithm is resumed in Algorithmus 3

7.2 ON THE CHOICE OF GOOD PENALTY PARAMETERS

Unlike the previous algorithms, in this one we have two penalty parameters: the first is considered when the hybrid algorithm is running in convex mode, while the second is considered right after the switch to non-convex mode.

The best choice for c^{co} corresponds to the best c value found when running the ADMM-SF algorithm. This value must provide a fast decrease of the target function, as well as the RMSE curve.

For what concern the second parameter, c^{nc}, it is not true that the best choice corresponds to what obtained from the running of the ADMM algorithm. This is caused by the fact that in the hybrid approach, the non-convex problem does not start from the initial zero

for $t = 0$ to ∞ **do**
  if $t = 0$ **then**
    Initialize local position $\hat{\mathbf{x}}_i^0$
    Build messages $\mathbf{m}_i^0 = \mathbf{A}_i \hat{\mathbf{x}}_i^0$
  else
    if $t < t^{sw}$ **then**
      Update local positions $\hat{\mathbf{x}}_i^t$ via (62) using convex function (73)
    else
      Update local positions $\hat{\mathbf{x}}_i^t$ via (62) using non-convex function (88)
    end
    Build messages $\mathbf{m}_i^t = \mathbf{A}_i \hat{\mathbf{x}}_i^t + \boldsymbol{\lambda}_i^{t-1} / c_i^{t-1}$
  end
  $\Rightarrow$ Broadcast values $\mathbf{m}_{\cdot,i,j}^t$ to region $j \in \mathcal{N}_i$
  $\Leftarrow$ Receive values $\mathbf{m}_{\cdot,j,i}^t$ from region $j \in \mathcal{N}_i$
  Evaluate the mixing values $\mathbf{z}_i^t$ via (68)
  if $t = 0$ **then**
    Reset memory $\boldsymbol{\lambda}_i^0 = \mathbf{0}$
  else
    Update memory $\boldsymbol{\lambda}_i^t = \boldsymbol{\lambda}_i^{t-1} + c_i^{t-1} \left(\mathbf{A}_i \hat{\mathbf{x}}_i^t - \mathbf{z}_i^t \right)$
  end
  Evaluate $\Gamma_i^t = \left|\left| \mathbf{A}_i \hat{\mathbf{x}_i}^t - \mathbf{z}_i^t \right|\right|$
  if $\Gamma_i^t < \tau$ **then**
    $t^{sw} = t$
  end
  if $t = t^{sw}$ **then**
    $c_i^t = c^{nc}$
  else
    if $t > t^{sw}$ **then**
      Update c_i^t value via (94)
    else
      $c_i^t = c^{co}$
    end
  end
  Evaluate $\mathbf{y}_i^t = \mathbf{D}_i^{-1} \mathbf{A}_i^\top \left(\mathbf{z}_i^t - \boldsymbol{\lambda}_i^t / c_i^t \right)$
end

Algorithmus 3 : Hybrid ADMM algorithm which exploits convex and non-convex problems.

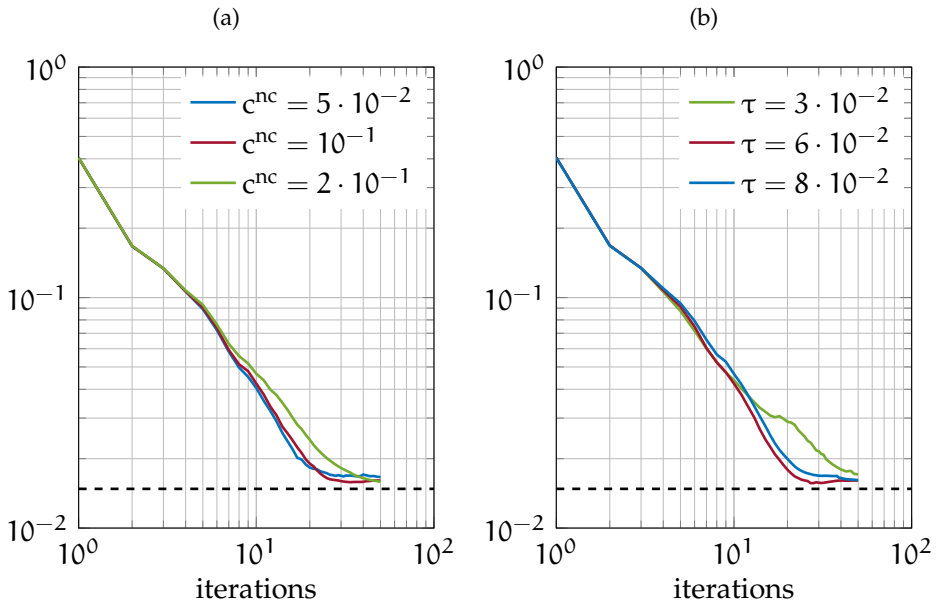


Figure 19: RMSE performances of hybrid ADMM algorithm on 500 nodes network. (a) Simulation with fixed threshold value $\tau = 5 \cdot 10^{-2}$ and different values of c^{nc}; (b) Simulation with fixed $c^{nc} = 10^{-1}$ value and different thresholds.

value for the nodes positions, but it starts from a raw solution that needs to be refined.

In the following section we show how the algorithm performance are affected by the choice of these parameters.

7.3 NUMERICAL SIMULATIONS

We now show some simulation results on our test networks. Like in the other chapters, we will perform numerical simulations on these three networks:

- 40 nodes network, of which 10 are anchor nodes;
- 500 nodes network, of which 10 are anchor nodes;
- 1000 nodes network, of which 20 are anchor nodes.

Some of the algorithm parameters are fixed, in particular they are $\delta_c = 1.01$, $\theta_c = 0.98$, $\lambda_{max} = 10^4$.

The value of c^{co} is as well fixed to the best values found in Chapter 5 for each considered network.

In Figure 19a we can see how the non-convex penalty parameter c^{nc} affects the RMSE performance of the algorithm. In particular by varying its value we can improve the RMSE value that the algorithm can achieve at convergence. By performing different simulations we found that a good value for the 500 nodes network is $c^{nc} = 10^{-1}$.

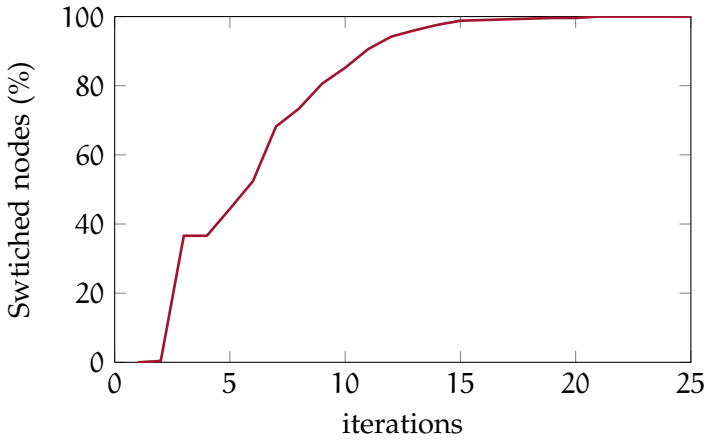


Figure 20: 500 nodes network. Amount of nodes (in percentage) that have switched to the non-convex mode, for each iteration.

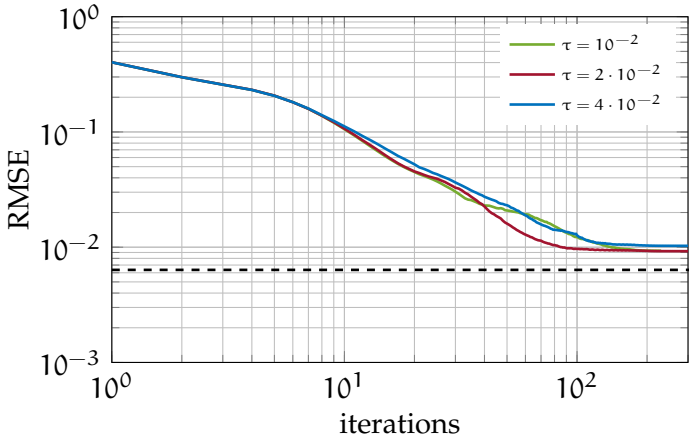


Figure 21: Hybrid ADMM approach on the 1000 nodes network for different threshold values. The non-convex penalty parameter is fixed to $c_{nc} = 2 \cdot 10^{-2}$.

At this point we can try to shift the threshold in order to improve the performance. In Figure 19b we can see that increasing the threshold value to $\tau = 6 \cdot 10^{-2}$, the algorithm converges faster. With $\tau = 3 \cdot 10^{-2}$ the algorithm converges in more than 50 iterations while we can reach the same result in only 20 iterations, by choosing a good threshold parameter. This implies that the choice of a good threshold value is fundamental in order to have a fast convergence.

In Figure 20 we can see how the nodes switch from the convex to the non-convex mode while the algorithm execution proceeds.

We can do the same analysis in the 1000 nodes network, in particular we can see how the threshold affects the RMSE performances. After choosing a good value for the non-convex penalty parameter $c^{nc} = 2 \cdot 10^{-2}$, we can see in Figure 21 how $\tau = 2 \cdot 10^{-2}$ may be a good threshold value.

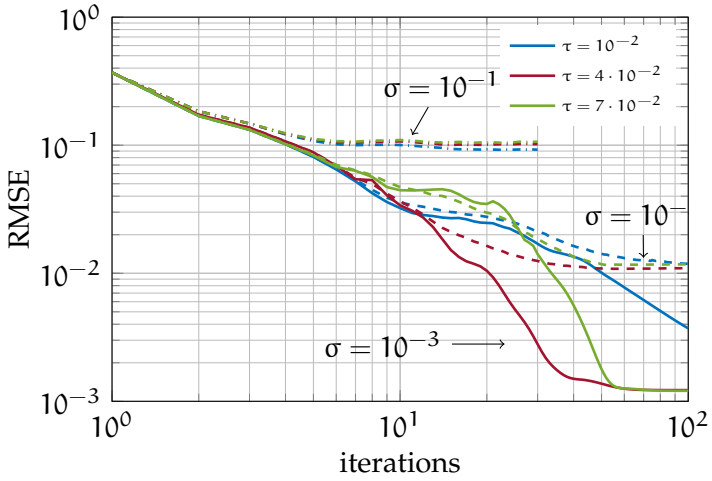


Figure 22: Average RMSE performances of Hybrid ADMM approach on the 40 nodes network for different threshold values. The non-convex penalty parameter is fixed to $c^{nc} = 5 \cdot 10^{-2}$.

Finally we consider the 40 nodes network. Like in the previous chapters we consider 50 noise realizations for each noise level. After choosing a good penalty parameter, $c^{nc} = 5 \cdot 10^{-2}$, we can look to the performances for different threshold values. The RMSE performances for three different threshold are shown in Figure 22. The best threshold is $\tau = 4 \cdot 10^{-2}$ that allows the algorithm to converge in only 40 iterations.

8

NUMERICAL SIMULATIONS AND PERFORMANCES COMPARISONS

In this chapter we will compare the performances of the algorithms discussed in this thesis work. The considered algorithms are:

- *SF*: It is the convex relaxation approach proposed by [24] and discussed in Chapter 3;
- *ADMM-SF*: It is the SF problem implemented in an ADMM fashion as discussed in Chapter 5;
- *ADMM*: It is the original non-convex localization problem solved in an ADMM fashion, as proposed by [11] and discussed in Chapter 6;
- *ADMM-H*: It is the hybrid algorithm proposed in Chapter 7;
- *SDP*: It is the Semi Definite Programming approach proposed by [23].

In order to test the performances of these algorithms we consider three different networks:

- A network composed of 40 nodes, of which 10 are anchor nodes. Since this network is small, we consider different noise realizations for each investigated noise level. In particular we consider three standard deviation cases: $\sigma = 10^{-1}$, $\sigma = 10^{-2}$ and $\sigma = 10^{-3}$, in order to understand how these algorithms behave with low and high noise levels;
- A network composed of 500 nodes, of which 10 are anchor nodes. In this case we consider a single realization of the noise, that provides a noise standard deviation of $\sigma = 2 \cdot 10^{-2}$;
- A network composed of 1000 nodes, of which 20 are anchor nodes. The noise standard deviation is $\sigma = 7 \cdot 10^{-3}$.

As a measure of the performance we will provide the RMSE, which is calculated as

$$\text{RMSE} = \frac{\mathrm{E}\left[\left|\left|\mathbf{x}^t - \mathbf{x}\right|\right|\right]}{\sqrt{N}} \tag{103}$$

where $\mathbf{x}^t$ are the computed nodes positions at iteration t, $\mathbf{x}$ are the real positions of the nodes and N is the number of nodes in the network.

We will also provide measurements on the algorithm complexity, considering the mean and maximum execution time at each iteration.

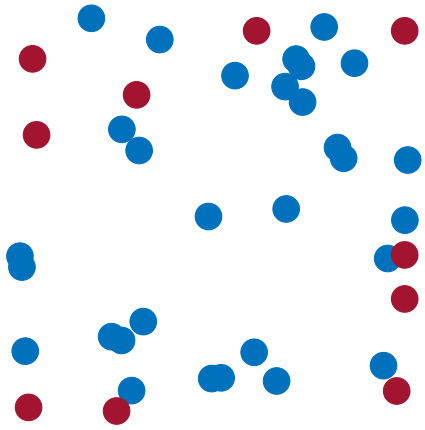


Figure 23: Network composed of 40 nodes, of which 10 (indicated by red dots) are anchor nodes.

ALGORITHM	c^{co}	c^{nc}	τ	θ_c	δ_c	λ_{max}
SF	-	-	-	-	-	-
ADMM-SF	$5 \cdot 10^{-3}$	-	-	0.98	1.01	10^4
ADMM	-	$5 \cdot 10^{-2}$	-	0.98	1.01	10^4
ADMM-H	$5 \cdot 10^{-3}$	$5 \cdot 10^{-2}$	$4 \cdot 10^{-2}$	0.98	1.01	10^4

Table 2: Algorithms parameters for the 40 nodes network test

Defining with T_i^t the time required by node i at iteration t to run the algorithm, we can measure the complexity of the algorithm providing the mean execution time at iteration t

$$\bar{T}^t = E\left[T_i^t\right] \tag{104}$$

and maximum execution time at iteration t

$$T_{max}^t = \max_i T_i^t \tag{105}$$

8.1 40 NODES NETWORK

We consider a network composed by 40 nodes, distributed in a square of unit size, like in Figure 23. As we can see, there are 10 anchor nodes which are basically situated on the external part of the network. Each node can communicate with nodes that are located in a radius of 0.3. We consider 50 noise realizations for each one of the noise standard deviations.

All the algorithms that we are testing are set with the best parameters, that we found in the previous chapters and are resumed in Table 2.

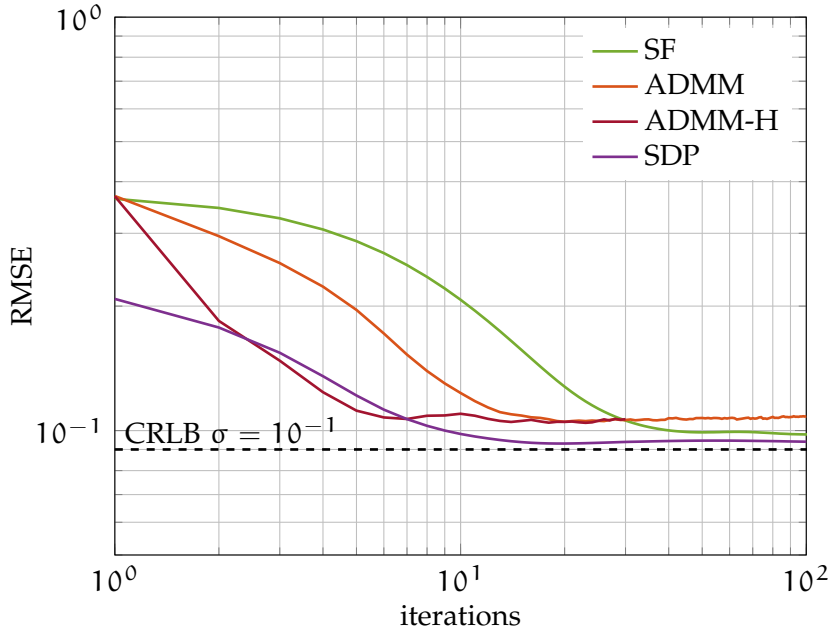


Figure 24: 40 nodes network with noise standard deviation $\sigma = 10^{-1}$. RMSE performances provided by the discussed algorithms.

In Figure 24 we see what happen when we are dealing with high noise measurements. In this case the noise level is $\sigma = 10^{-1}$.

All the algorithms are able to converge to an RMSE value which is tight to the CRLB (indicated by the dashed line). In this case, We do not show the performance of the ADMM-SF algorithm because it is the same of the ADMM-H algorithm.

We note that the SF curve converges after 40 iterations, while the ADMM one converges after 15 iterations, meaning that the approach proposed in Chapter 3 makes sense. A significant improvement is done by the ADMM-H algorithm that converges in only 5 iterations. The SDP algorithm converges in a little more than 10 iterations to an RMSE value which is very tight to the bound.

In Figure 25 we can see the performances of the five algorithms in the $\sigma = 10^{-2}$ and $\sigma = 10^{-3}$ cases. The SF performance in these two noise levels is almost the same and the algorithm converges in 100 iterations to an RMSE value which is far from the CRLB. In particular we can note how the RMSE value reached by SF in the $\sigma = 10^{-3}$ is very high compared to the value provided by the bound.

The ADMM-SF algorithm improves, obviously, only the number of iterations required to converge as we can note from the fact that it always converges to the RMSE value provided by SF, but faster.

An improvement on the RMSE value is given, instead, by the ADMM algorithm, that by solving the original non-convex problem, achieves RMSE values which are, in both cases, close to the bounds. In particular, in the $\sigma = 10^{-2}$ case the algorithm converges in 45 iterations,

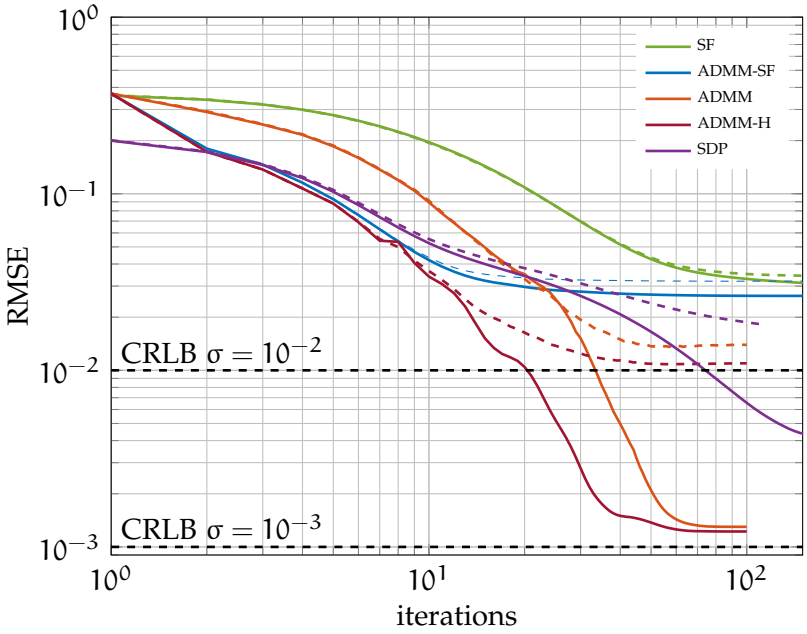


Figure 25: 40 nodes network. RMSE performances provided by the discussed algorithms. Dashed lines indicate the results for $\sigma = 10^{-2}$ while solid lines indicate the results for $\sigma = 10^{-3}$.

ALGORITHM	c^{co}	c^{nc}	τ	θ_c	δ_c	λ_{max}
SF	-	-	-	-	-	-
ADMM-SF	$2 \cdot 10^{-3}$	-	-	0.98	1.01	10^4
ADMM	-	$5 \cdot 10^{-2}$	-	0.98	1.01	10^4
ADMM-H	$2 \cdot 10^{-3}$	10^{-1}	$6 \cdot 10^{-2}$	0.98	1.01	10^4

Table 3: Algorithms parameters for the 500 nodes network test

while in the $\sigma = 10^{-3}$ case the algorithm converges in more than 60 iterations.

Finally, the proposed hybrid approach, indicated in the figure as ADMM-H, provides the best results. For $\sigma = 10^{-2}$ the algorithm converges in 30 iterations to an RMSE value which is very close to the bound, while for $\sigma = 10^{-3}$ it converges in a little more than 40 iterations.

Even the performances of the SDP approach [23] are indicated in the figure and we can note that they are worse than what achieved by using the ADMM and ADMM-H algorithms.

8.2 500 NODES NETWORK

In Figure 27 we can see the performances of the four discussed algorithms, when performing localization on the 500 nodes network[26]

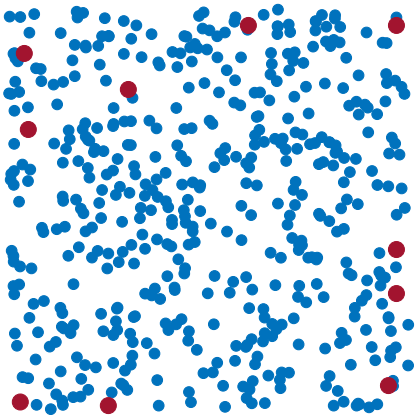


Figure 26: Network composed of 500 nodes, of which 10 (indicated by red dots) are anchor nodes.

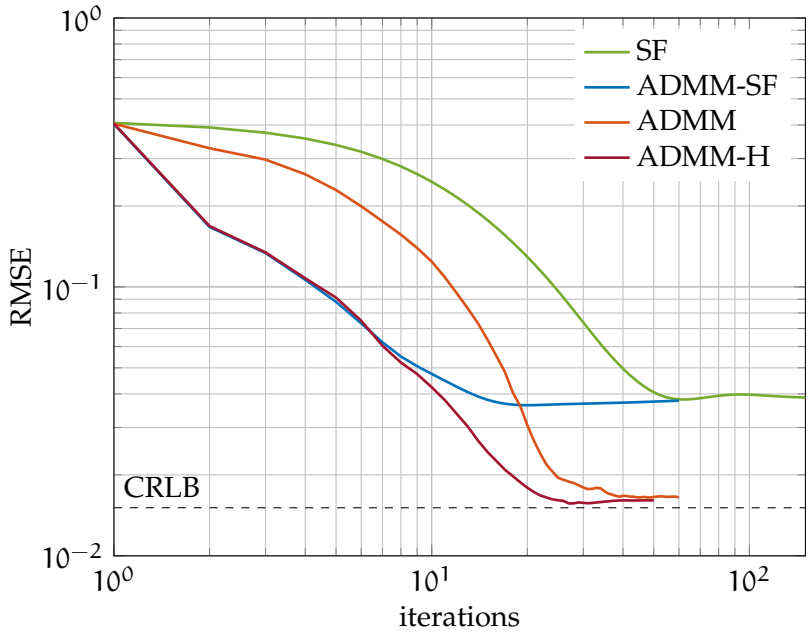


Figure 27: 500 nodes network. RMSE performance of discussed algorithms.

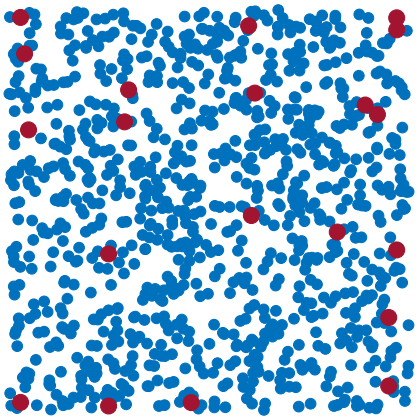


Figure 28: Network composed of 1000 nodes, of which 20 (indicated by red dots) are anchor nodes.

ALGORITHM	c^{co}	c^{nc}	τ	θ_c	δ_c	λ_{max}
SF	-	-	-	-	-	-
ADMM-SF	10^{-3}	-	-	0.98	1.01	10^4
ADMM	-	$2 \cdot 10^{-2}$	-	0.98	1.01	10^4
ADMM-H	10^{-3}	$2 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	0.98	1.01	10^4

Table 4: Algorithms parameters for the 1000 nodes network test

shown in Figure 26. All the tested algorithms are set with parameters indicated in Table 3.

From Figure 27 we note that the SF algorithm requires 50 iterations to converge while the ADMM-SF solution can achieve the same RMSE result in only 13 iterations.

Considering the ADMM approach we can see that its RMSE performance curve at convergence is close to the result provided by the CRLB and the algorithm converges in 35 iterations.

The best result is given by the ADMM-H algorithm that provides, in only 25 iterations, a result which is slightly better than the result given by ADMM.

8.3 1000 NODES NETWORK

Like in the previous testing network, the nodes are located in a square of unit size, but in this case there are 20 anchor nodes, as indicated in Figure 28.

Performances on this network, are shown in Figure 29, where the four algorithms are set with parameters indicated in Table 4.

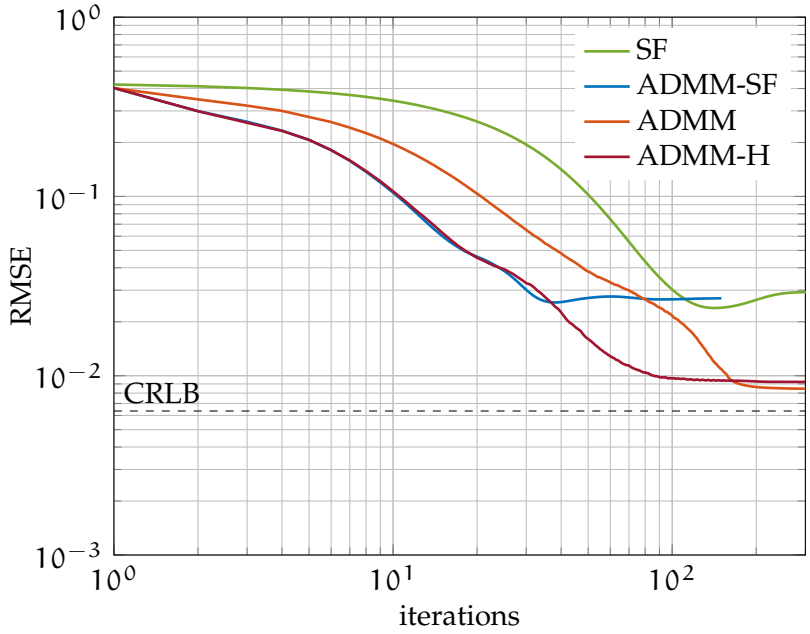


Figure 29: 1000 nodes network. RMSE performance of the discussed algorithms.

Initially we note that, like in the other networks, the worst localization algorithm in terms of RMSE performance, is the SF algorithm, which converges after more than 100 iterations.

An important improvement is given by the ADMM-SF solution, which achieves in only 30 iterations, the same RMSE value provided by SF. To get a better RMSE value, instead, we have to opt for the ADMM algorithm which is slower to converge but can achieve a RMSE value which is closer to the CRLB.

Finally, the best performance is provided by the hybrid approach, which converges to the same value provided by ADMM, but in only 80 iterations.

8.4 TIME COMPLEXITY

It is interesting at this point, to look at the complexity of the considered algorithms, in terms of execution times. To do this, we consider the two measures described by (104) and (105).

The first measure represents the average time over sensor nodes, at iteration t. The second, which is more meaningful, is the maximum execution time required by sensor nodes, at iteration t. This value is particularly interesting, considering that we are dealing with distributed algorithms that work in a parallel fashion.

In Figure 30 we compare the average times (dashed lines) and the maximum times (solid lines) of SF, ADMM-SF and ADMM-H algo-

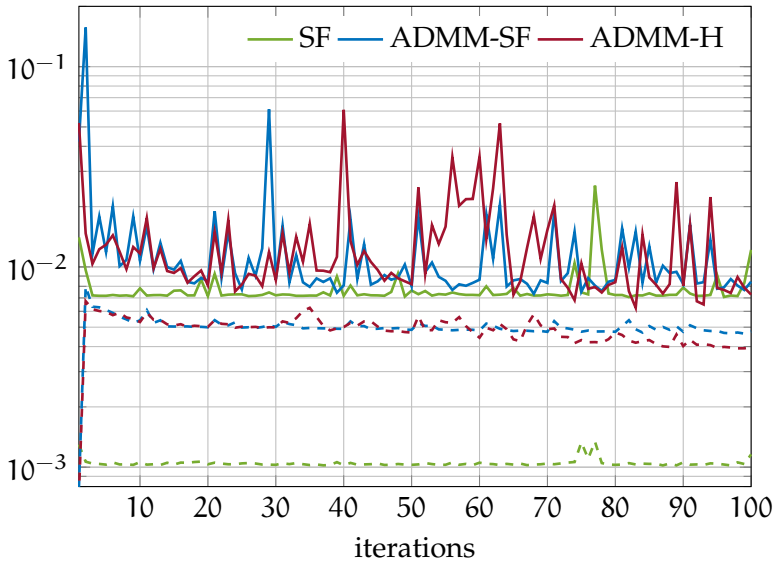


Figure 30: Average time at iteration t, $\bar{T}^t$ (dashed lines) and maximum time at iteration t, T^t_{max} (solid lines) measured on the 500 nodes network.

rithms, measured when performing localization on the 500 nodes network.

From the figure we note that the SF algorithm is the faster, in mean. This may seem a good thing, compared to the other approaches, but we must consider that when dealing with a distributed algorithm, we cannot forget the communication costs. It is true that the SF iterations are fast in terms of execution time, but the SF approach requires a lot of iterations in order to converge. This means that it requires a lot of data exchange between nodes and consequently more energy usage.

Finally, we note that the values of $\bar{T}^t$ and T^t_{max}, for the ADMM-H and the ADMM algorithms are very similar and $\bar{T}^t$, in both cases, decreases after the algorithm convergence.

The value of T^t_{max} is very similar between the considered algorithms. This is an important parameter in order to estimate the algorithm performances since it indicates the maximum time required by a node in order to execute its job at each iteration. Since we are dealing with a parallel localization algorithm, each node can proceed with its iterations after receiving data from all its neighbors, so this value represents a bottleneck on the execution time.

8.5 ROBUSTNESS OF THE ALGORITHM

An interesting evaluation can be done on the robustness of the discussed algorithms for what concern the choice of the c penalty parameter. We derived and showed the optimal values found for each algorithm. The optimal value of this parameter is related to the network characteristics (e.g., the network size or the connectivity degree).

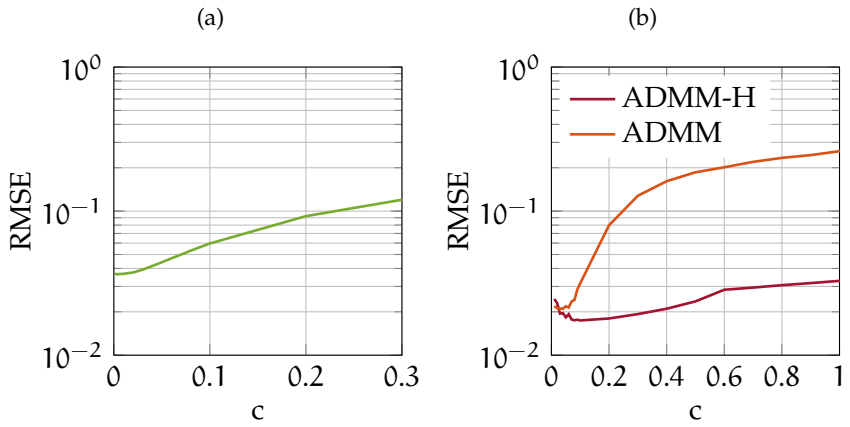


Figure 31: Achieved RMSE value at the 20th iteration when running localization algorithms with different penalty parameter values. a)ADMM-SF algorithm b) ADMM and ADMM-H algorithms. In the ADMM-H algorithm c represents the value of the non-convex mode penalty parameter c^{nc}.

If this value must be chosen very accurately in order to achieve a good RMSE value, this may be a problem, considering that we usually don't have a-priori knowledge on the network characteristics. We now want check for the robustness of these algorithms and in order to do this, we run them by varying the value of c and by setting a maximum number of 20 iterations. This allows to see how much the RMSE value changes by varying the value of the penalty parameter. From Figure 31 we can see that ADMM-H is the most robust of the considered algorithms. In particular, the ADMM algorithm has a big variability of the achieved RMSE value for low values of c. This happens in all the algorithms but ADMM-H appears to be more robust since the variability of the achieved RMSE value is small and this means that it can achieve a good RMSE value also if it is set with a non-optimal penalty parameter.

8.6 ACHIEVED PERFORMANCE IN A MOVING NETWORK

It is interesting to see if the proposed ADMM-H algorithm is able to provide significant results also when the nodes of the network are moving. To test this we consider the 40 nodes network with range measurements affected by Gaussian noise with zero mean and standard deviation $\sigma = 10^{-2}$. In this simple example the nodes speed on the x and y axes is modeled as $\mathcal{N}\left(10^{-3}, 5 \cdot 10^{-3}\right)$ and it is maintained constant over the time. The testing scenario is shown in Figure 32a where we can see the path followed by each node. The empty circles show the initial positions of network nodes, while the filled circles show the final positions. From the figure we can also note that anchor nodes are not moving.

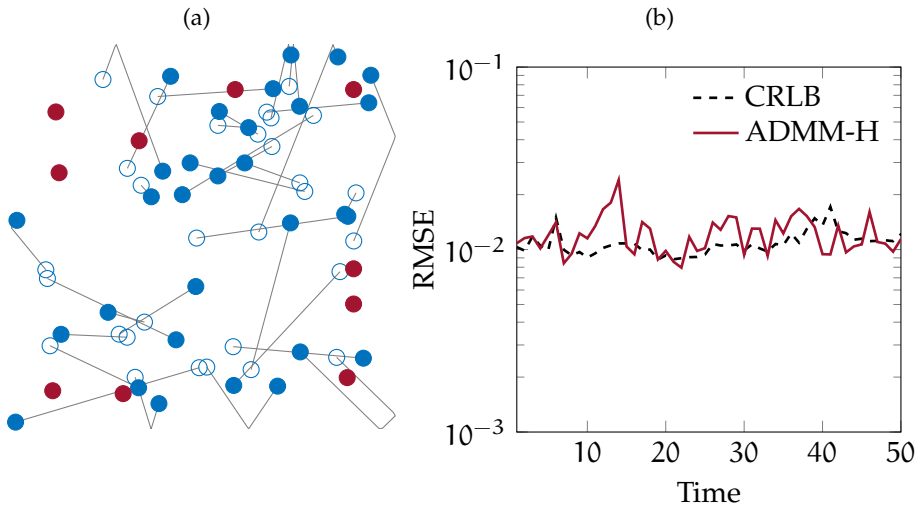


Figure 32: RMSE performance of the ADMM-H algorithm in a moving network. (a) Empty circles represent the initial positions of nodes, while filled circles are the final positions, at time instant 50. Red circles are anchor nodes. (b) RMSE value achieved at iteration 30 for each time instant.

From Figure 32b we can see that the CRLB changes over the time, since the network nodes are moving and this affects its characteristics, e.g., the nodes connectivity. The figure also shows the RMSE value achieved by the ADMM-H algorithm at each time instant. We can see that the computed RMSE values *follow* the bound provided by the CRLB and this means that the algorithm can successfully be used for tracking applications.

8.7 CONCLUSIONS ON NUMERICAL SIMULATIONS

From these simulations we can see that the ADMM-SF algorithm, proposed in this thesis work, is a good way to improve the performance of the SF algorithm, since in all the considered network, it achieves better results in terms of iterations. From our simulations the ADMM-SF approach requires to converge from 10% to 30% of the number of iterations needed by SF.

The second significant comparison is between the ADMM algorithm and the proposed hybrid approach, ADMM-H, since they generally achieve the same RMSE value at convergence. From our simulations, the ADMM-H algorithm can need only the 50% of the iterations required by ADMM to converge, like seen in the 1000 nodes network.

9 CONCLUSIONS

In this thesis we introduced one of the most important problems in Wireless Sensor Networks: localization. This problem can in fact be solved in different ways. We saw how it is possible to exploit ranging measurements, available at sensor nodes, to develop distributed algorithms that work in a parallel fashion, in order to estimate the positions of each node of the network.

We saw that the algorithm proposed in the work by Soares, Xavier, and Gomes [24] can be improved, in terms of number of iterations needed to converge, by using the ADMM approach described in Chapter 4. This allows to decompose the localization problem into smaller local sub-problems solved in each node of the network. The improvement provided by ADMM is although not sufficient if we want to reach low RMSE values, since the solution provided by the convex approach is typically not too precise.

The second approach we investigated, proposed in the work by Erseghe [11], solves the original non-convex problem in an ADMM fashion and we saw that this method provides good RMSE values, which, in most cases, can reach the bound provided by CRLB.

Finally, we introduced an hybrid ADMM approach which starts by solving the convex problem, exploiting its fast convergence speed, and then refines the achieved solution by switching to the optimization of the original non-convex problem. From the numerical simulations shown in Chapter 8 we found that the proposed approach reduces the number of iterations required to reach convergence in all the cases but the improvement is bigger when dealing with large networks. This solution is robust on the choice of the penalty parameter and can also be used for tracking applications.

BIBLIOGRAPHY

[1] Roberto Andreani, Ernesto G Birgin, José Mario Martínez, and María Laura Schuverdt. "On augmented Lagrangian methods with general lower-level constraints." In: *SIAM Journal on Optimization* 18.4 (2007), pp. 1286–1309.

[2] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[3] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[4] Ernesto G Birgin and José Mario Martínez. *Practical augmented Lagrangian methods for constrained optimization*. Vol. 10. SIAM, 2014.

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers." In: *Foundations and Trends® in Machine Learning* 3.1 (2011), pp. 1–122.

[6] Ka Wai Cheung and Hing-Cheung So. "A multidimensional scaling framework for mobile location using time-of-arrival measurements." In: *Signal Processing, IEEE Transactions on* 53.2 (2005), pp. 460–470.

[7] Fan RK Chung. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

[8] Juan Cota-Ruiz, Jose-Gerardo Rosiles, Pablo Rivas-Perea, and Ernesto Sifuentes. "A distributed localization algorithm for wireless sensor networks based on the solutions of spatially-constrained local problems." In: *Sensors Journal, IEEE* 13.6 (2013), pp. 2181–2191.

[9] Benoit Denis, Jean-Benoît Pierrot, and Chadi Abou-Rjeily. "Joint distributed synchronization and positioning in UWB ad hoc networks using TOA." In: *Microwave Theory and Techniques, IEEE Transactions on* 54.4 (2006), pp. 1896–1911.

[10] Giuseppe Destino and Giuseppe Abreu. "On the maximum likelihood approach for source and network localization." In: *Signal Processing, IEEE Transactions on* 59.10 (2011), pp. 4954–4970.

[11] Tomaso Erseghe. "A Distributed and Maximum-Likelihood Sensor Network Localization Algorithm Based Upon a Nonconvex Problem Formulation." In: *Signal and Information Processing over Networks, IEEE Transactions on* 1.4 (2015), pp. 247–258.

[12] Daniel Gabay and Bertrand Mercier. "A dual algorithm for the solution of nonlinear variational problems via finite element approximation." In: *Computers & Mathematics with Applications* 2.1 (1976), pp. 17–40.

[13] R Glowinski and A Marocco. "Sur l'approximation par elements finis d'ordre un, et la resolution part penalisation-dualite, d'une class de problems de Dirichlet non lineaires'." In: *Rev. Francaise Automat, informat. Recherche Ooperationalle Ser. Rouge Anal. Numer* (), pp. 4–2.

[14] JB Hiriart-Urruty and C Lemarechal. "Convex Analysis and Minimization AlgorithmsSpringer." In: *New York, NY* (1993).

[15] Koen Langendoen and Niels Reijers. "Distributed localization in wireless sensor networks: a quantitative comparison." In: *Computer Networks* 43.4 (2003), pp. 499–518.

[16] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. "Semidefinite relaxation of quadratic optimization problems." In: *IEEE Signal Processing Magazine* 27.3 (2010), p. 20.

[17] Olvi L Mangasarian, Robert R Meyer, and Stephen M Robinson. *Nonlinear Programming 3: Proceedings of the Special Interest Group on Mathematical Programming Symposium Conducted by the Computer Sciences Department at the University of Wisconsin–Madison, July 11-13, 1977*. Academic Press, 2014.

[18] Guoqiang Mao, Barış Fidan, and Brian DO Anderson. "Wireless sensor network localization techniques." In: *Computer networks* 51.10 (2007), pp. 2529–2553.

[19] Yurii Nesterov. "A method of solving a convex programming problem with convergence rate O (1/k2)." In: *Soviet Mathematics Doklady*. Vol. 27. 2. 1983, pp. 372–376.

[20] Neal Patwari, Joshua N Ash, Spyros Kyperountas, Alfred O Hero III, Randolph L Moses, and Neiyer S Correal. "Locating the nodes: cooperative localization in wireless sensor networks." In: *Signal Processing Magazine, IEEE* 22.4 (2005), pp. 54–69.

[21] Theodore S Rappaport et al. *Wireless communications: principles and practice*. Vol. 2. Prentice Hall PTR New Jersey, 1996.

[22] Qingjiang Shi, Chen He, Hongyang Chen, and Lingge Jiang. "Distributed wireless sensor network localization via sequential greedy optimization algorithm." In: *Signal Processing, IEEE Transactions on* 58.6 (2010), pp. 3328–3340.

[23] Andrea Simonetto and Geert Leus. "Distributed maximum likelihood sensor network localization." In: *Signal Processing, IEEE Transactions on* 62.6 (2014), pp. 1424–1437.

[24] Cláudia Soares, João Xavier, and João Gomes. "Simple and fast convex relaxation method for cooperative localization in sensor networks using range measurements." In: *Signal Processing, IEEE Transactions on* 63.17 (2015), pp. 4532–4543.

[25] Zizhuo Wang, Song Zheng, Yinyu Ye, and Stephen Boyd. "Further relaxations of the semidefinite programming approach to sensor network localization." In: *SIAM Journal on Optimization* 19.2 (2008), pp. 655–673.

[26] Y. Ye. *Computational Optimization Laboratory*. URL: http://www.stanford.edu/~yyye/Col.html.