ISEE.U: Distributed estimation and control for improved target localization accuracy

Miguel Araújo Vasques

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Cláudia Alexandra Magalhães Soares
Prof. João Pedro Castilho Pereira Santos Gomes

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Cláudia Alexandra Magalhães Soares
Member of the Committee: Prof. Pedro Manuel Urbano de Almeida Lima

November 2017

Acknowledgments

To my supervisor, Prof. Cláudia Soares, I would like to express my profound gratitude for all the guidance and teaching that she provided throughout this work. The excellent guidance that was provided was vital to the success of this work. To Prof. João Pedro Gomes, thank you for all the help that you gave me on this work.

To my mother, father and brother and the rest of my family, thank you for all the support and for always believing in me and for making me reach for bigger goals. To Mafalda, thank you for never letting me quit and being by my side on this journey. Lastly, to my friends, thank you for always being there for me and for reminding me everyday that life is not just work.

Abstract

We propose two distributed algorithms to localize multiple static or moving targets using a network of agents from range measurements and to find the control for the network that increases the accuracy of localization of those targets. One of them is based on a tighter approximation, but is biased, while the other is unbiased. We named the latter as ISEE.U.

In both algorithms, each agent evaluates its position in the network using a local estimate of the Fisher Information Matrix. We note that the FIM is decomposable through agents and, from this key fact, each agent can minimize the overall estimator's error ellipsoid volume and establish its next movement. Target position estimates are also computed in the same distributed process. We build on two linear approximations to the target localization problem and we get, at each iteration of our consensus-like method, both a FIM estimate and a position estimate, from which we perform estimation and control, even for sparsely connected networks.

Simulation results showed that the ISEE.U estimate could be improved especially in the onset of the operation and, thus, we consider refinement methods to be used on the first few moments of the active localization procedure.

ISEE.U was then compared against a state-of-the-art method in different scenarios showing similar results and sometimes even outperforming the benchmark, and using x100 less computation time, even when ISEE.U is running in one central CPU.

Keywords

Active estimation, distributed estimation, distributed control, target localization, Fisher Information Matrix.

Resumo

São propostos dois algoritmos distribuídos para localizar múltiplos alvos estáticos ou em movimento usando uma rede de agentes, a partir de medidas de alcance, e encontrar o controlo para a rede que aumenta a exatidão de localização desses alvos. Um deles é baseado numa aproximação mais rigorosa, mas é enviesado, enquanto que o outro não o é. Nós chamamos ao segundo ISEE.U.

Em ambos os algoritmos, cada agente avalia a sua posição na rede usando uma estimativa local da Matriz de Informação de Fisher. A MIF é decomposta pelos agentes e, a partir deste facto fundamental, cada agente pode minimizar o volume do elipsoide de erro geral e estabelecer o seu próximo movimento. As estimativas da posição dos alvos são também calculadas no mesmo processo distribuído. Nós partimos de duas aproximações lineares para o problema de localização e obtemos, em cada iteração do nosso método semelhante ao consensos, uma estimativa da MIF e da posição do alvo, a partir dos quais nós fazemos estimação e controlo, até para redes que são pouco ligadas.

Os resultados das simulações mostraram que a estimativa ISEE.U podia ser melhorada, especialmente no início da operação e, por isso, nós consideramos métodos de refinamento para serem usados nas primeiras iterações do procedimento de localização.

O ISEE.U foi depois comparado com um método da literatura em diferentes cenários nos quais mostrou resultados semelhantes e por vezes até melhores que o benchmark, usando x100 menos tempo de computação, mesmo quando o ISEE.U corre num CPU central.

Palavras Chave

Estimação ativa, estimação distribuída, controlo distribuído, localização de alvos, Matriz de Informação de Fisher.

Contents

List of Figures

List of Tables

Abbreviations

BB *Barzilai-Borwein*

BP *Belief propagation*

CDF *Cumulative Distribution Function*

CRLB *Cramer-Rao Lower Bound*

FIM *Fisher Information Matrix*

FoV *Field of View*

GPS *Global Positioning System*

MAE *Mean Absolute Error*

ML *Maximum Likelihood*

MMSE *Minimum Mean Square Error*

MVU *Minimum Variance Unbiased*

pdf *probability density function*

RMSE *Root-Mean-Square Error*

R-LS *Range-Based Least Squares*

SR-LS *Squared-Range-Based Least Squares*

SNR *Signal to Noise Ratio*

USR-LS *Unconstrained Squared-Range-Based Least Squares*

Chapter 1

Introduction

1.1 Motivation

The localization problem has been recurrent in the history of mankind. At first people had the need to locate themselves discovering multiple ways to approach the problem. One solution and arguably the simplest one is to use landmarks with known absolute positions around us and average them to estimate our own position. Then, multiple instruments were developed to aid in the self localization process, like maps and compasses, but since then technological development made it possible that almost everyone has everyday access to a device that can autonomously locate itself.

The most frequently used method to perform self localization is the *Global Positioning System* (GPS), which uses a network of satellites and trilateration to find the position of a receiver. Unfortunately, GPS requires clear line-of-sight with at least 4 satellites at the same time instant. This limitation confines its use to outdoor, clear sky usage.

Other localization problem that has been around for a long time is how can an object or a missing person be found. This is a lot more difficult to address since we may not have much information about what we want to find. One common solution is the trial and error approach. With the development of technology, many people tried to address this problem in ways that could save people's efforts and time. One could try using GPS, but this means it would be necessary to implement a GPS receiver (and a communications transceiver) in every object, and we know that GPS cannot operate on most environments. GPS-denied environments, especially indoor scenarios, are a major concern because they are frequent and even the average person deals with the need for indoor localization everyday.

The problem also goes beyond that of a person trying to find a common household object. Several different economic sectors have to deal with this problem at a much larger scale, involving money and time. Examples of these sectors are:

- Logistics;
- Security;
- Minerals and oil exploration;

- Navigation;
- Wireless communications;
- Surveillance.

Nowadays, this problem is trending since there were major developments in autonomous vehicles that can be equipped with sensors and cooperate in networks to solve our problem.

Some instances of the localization problem in search/rescue applications are:

- A rescue mission in an hostile environment where the action of humans can be dangerous;
- The search for oil reserves underwater;
- Searching and handling packages in a large warehouse;
- Missions in other planets where the atmosphere is not suitable for humans;
- Search for land mines and enemy troops in a battlefield.

As you can see these are activities that are very difficult or impossible to be performed by humans and so it is necessary to create machines to do them. Motivated by these applications, we address a "canonical" problem where a team of mobile agents tries to locate a radiant source through range measurements, which can be obtained in various ways (e.g, as time of arrival of the signal or through the measure of power of the received signal [1]).

1.2 Problem

Our problem is basically how can a network of randomly deployed mobile agents accurately locate radiating sources and how can we control the movement of that network to better perform its task.

The easiest solution is the use of centralized networks. These are networks where every agent sends its measurements to a central node where the computations are made and then the central node reconfigures the network based on the collected data. This architecture brings some problems, mainly:

- The nodes are prone to failure and it is necessary to reconfigure all the network every time one fails or when more agents are added;
- The central node can also fail and if it does all the network has to stop until the central node is back online;
- A lot of time and energy can be wasted in communicating the data to the central node and in computational hardware for data processing.

So we have to find a more flexible solution that can easily adapt to different environments and network configurations without any intervention.

Another aspect of the problem is to find what control law to use to move the network. This is also very important because it will dictate the behavior of the network and ultimately improve the localization process. This control law also has to be general for all types of networks and environments to maintain the flexibility of our solution.

1.3 Overview of the Approach

The approach used here to solve the stated problem is to devise a network of agents where all the computations are made independently by each agent, without the need to report to a central node, by exchanging data only among neighbors (in terms of communication range). This is the definition we adopt for a distributed architecture.

We aim for a flexible and robust method that could easily adapt to different conditions with minimal effort from the user. Also, we want a solution that is easy to understand and also easy to implement.

The setup of our solution comprises (i) a network of mobile circular agents with a finite *Field of View* (FoV), meaning that the space where each agent can measure the position of the source and communicate with other agents is limited by a circle with finite range, and (ii) radiating sources which we will refer to as targets from now on. A visual representation of this setup is shown in Figure 1.1.

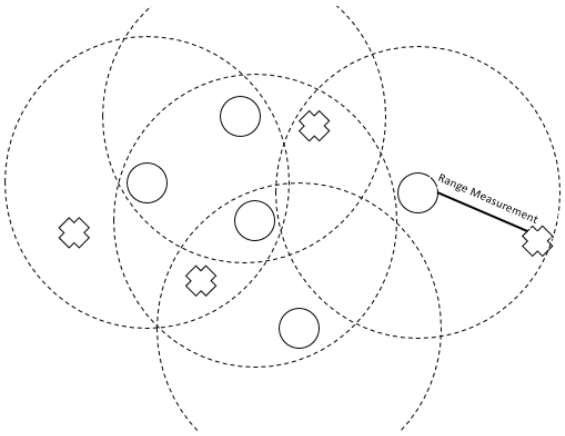


Figure 1.1: Setup for the considered solution. The circles represent the agents, the crosses represent the targets and the dashed lines are the agents' FoVs.

We will consider the localization of a radiating source, which we call target as an optimization problem. This optimization problem is formulated using a least-squares approach where we try to find the position of the target that minimizes a cost function that tries to fit the distance to the estimated target position with the measured noisy ranges. Two least-squares approaches can be used to solve this problem, *Range-Based Least Squares* (R-LS) and *Squared-Range-Based Least Squares* (SR-LS) [2]. Both approaches formulate optimization problems that are nonconvex, meaning that the corresponding cost function can have multiple locally optimal points and a optimal global solution is very difficult or even impossible to find. We can only try to simplify our optimization problem and find approximate solutions. For the R-LS a semidefinite relaxation is used but it is not guaranteed that the optimal value for it is the

same as the original R-LS problem. We use the SR-LS to formulate our optimization problem because according to [2], although being nonconvex, this problem can be efficiently and globally solved.

Now, it is possible to write this problem using a linear model and use the distributed linear estimator called ISEE.U, which is an approximation to the *Minimum Variance Unbiased* (MVU) linear estimator, to find an estimate for the position of the targets and the respective covariance matrices of those estimates. The ISEE.U estimator can be used by each agent to compute the position estimates through a consensus + innovations method where each agent exchanges information with its neighbors and uses it to compute the estimator, since this method approaches the centralized architecture of the problem. So now every agent of the network can compute an estimate for the target positions independently of any external entity. To the estimation phase of our method we call ISEE.Uest.

Finally we can use the covariance matrices that each agent can compute through the ISEE.U estimator to design the control law for the movement of the network. From the eigenvalues of the covariance matrix, which is the inverse matrix of the *Fisher Information Matrix* (FIM), we can compute the volume of an error ellipsoid for that target. The objective is to minimize the volume of that error ellipsoid in order to improve the accuracy of estimation of the target position. So the next position of the agent will always be the one that minimizes the volume of the error ellipsoid. To the control phase of our method we call ISEE.Uctl.

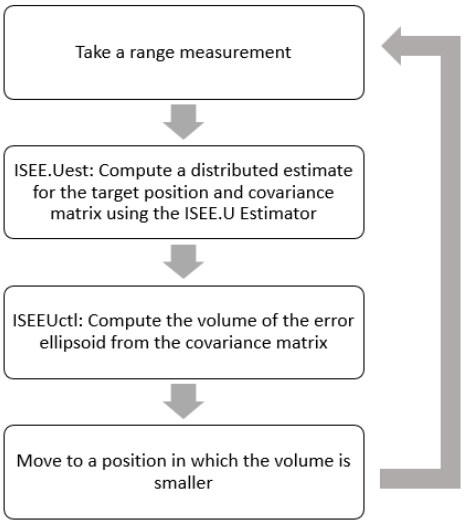


Figure 1.2: Diagram of the method for each agent and for each target.

1.4 Contributions

The main contributions provided by this thesis are:

- A distributed method to both estimate the target positions and control of the network agents;
- The distributed estimates and control are computed through a fast consensus + innovations scheme;
- A simple to implement, fast and flexible method demanding only range measurements to the target;

- A method that showed good performance when compared to a state-of-the-art algorithm using a particle filter approach.

Code available at `https://github.com/migvas/ISEE.U`.

1.5 Problem Formulation

We define a network composed by $n(t)$ agents and $K(t)$ targets. The position of agent i at time t is given by $\mathbf{s}_i(t) \in \mathbb{R}^d$, where $d = 2$ or $d = 3$ is the dimension of the space considered, and $\mathbf{p}_k(t) \in \mathbb{R}^d$ represents the position of target $k \in \{1, \ldots, K(t)\}$ at a discrete time instant t.

Consider also a graph given by $G(t) = (V(t), E(t))$ called the communications graph which represents the network established by the communications between agents and where $V(t) = \{1, ..., n(t)\}$ is the graph's set of nodes, where each node represents one agent, and $E(t) = \{i \sim j \,:\, i, j \in V(t)\}$ is the set of edges. Each edge represents a communication link between two agents of the network.

Finally, consider a graph given by $\mathcal{G}(t) = (\mathcal{V}(t), \mathcal{E}(t))$, the measurements graph, which represents the network of range measurements of the agents relative to the targets. The nodes of the graph are given by $\mathcal{V}(t) = V(t) \cup \mathcal{T}(t)$, where $\mathcal{T}(t) = \{1, ..., K(t)\}$ and where each node represents one agent or target of the network. In this graph the edges are given by $\mathcal{E}(t) = \{i \sim k \,:\, i \in V(t), k \in \mathcal{T}(t)\}$ where each edge is a range measurement between one agent and one target.

The range measurement obtained by agent i to target k at time t is given by

$$r_{ik}(t) = ||\mathbf{s}_i(t) - \mathbf{p}_k(t)|| + w_{ik}(t) \tag{1.1}$$

where $w_{ik}(t) \sim \mathcal{N}(0, \sigma^2)$ is a term of white Gaussian noise with zero mean and standard deviation σ. The noise will be analyzed in more detail in Section 3.1.

The data model is composed by the positions of the agents $\mathbf{s}_i(t)$ and by the respective range measurements $r_{ik}(t)$. The output is an estimate for the positions of the targets $\hat{\mathbf{p}}_k(t)$ and the control that defines the next positions for the agents $\mathbf{s}_i(t+1)$. A simple diagram representing the inputs and outputs of ISEE.U is presented in Figure 1.3.

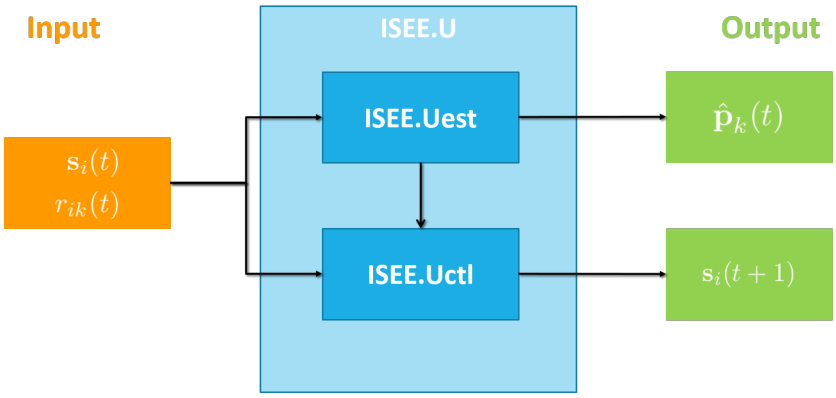


Figure 1.3: Visual representation of the inputs and outputs of ISEE.U.

1.6 Thesis Outline

This thesis is structured as follows: in Chapter 2 we present a set of methods found in the literature that tackle the localization problem in different ways and draw some comparisons to our method; in Chapter 3 a mathematically detailed explanation of our proposed method is given; in Chapter 4 some extensions to the proposed method are presented in order to make it more general and easily adaptable to different variations of our problem; in Chapter 5 we compute experimentally some important data for our method and present simulations that demonstrate its performance; in Chapter 6 we draw conclusions about the work developed and propose some future work that can be done in order to make it more complete.

Chapter 2

State of the Art

In this chapter we will present and analyze various methods found in the literature that address the localization problem described in Chapter 1.

We will also introduce and explain the the benchmark method [3] used for numerical comparisons in Section 5.4.

2.1 Literature Review

Some surveys [4],[5] already present taxonomies for a large number of methods that address the general target localization problem. Having our specific problem in mind, we classify the methods with the criteria in Figure 2.1.

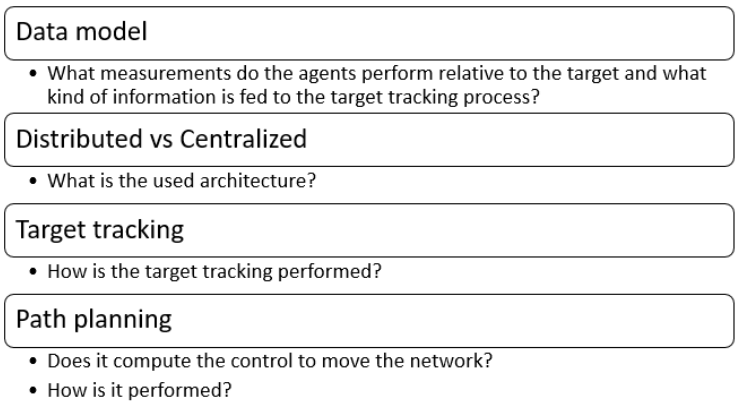


Figure 2.1: Criteria used to classify the selected active target localization methods.

These criteria are mere guidelines, since we face specific flavors of target localization and a variety of algorithms. This means that some methods consider a general approach of a sensor network performing target localization, like our proposal, while others consider more specific problems like one vehicle trying to locate one target. These methods that consider more restricted problems are also important to review

because sometimes they can be scaled and generalized.

We will start by reviewing the methods that perform the target tracking in a more generalized way using a network of multiple agents, as in our proposal, presented in [6],[7],[8],[9] and [10].

In [6] the problem consists in monitoring a set of mobile targets, in 2D, with a network of mobile robots equipped with omnidirectional sensors, which means that their FoV is circular. Each agent of the network can obtain noisy observations for the presence or absence of the target at a certain cell in its FoV but it does not try to locate the target. Instead, this method aims only to configure the network in order to maximize the monitoring performance for all targets. This is a similar problem to the one where we only want to cover the area around the target and not precisely locate it inside that area. So, the most important computations in this type of problems occur in the path planning phase, which aims to find the best control for the network in order to intercept the targets, maximizing their motorization time.

To intercept the mobile targets, the agents have to estimate which positions will be occupied by the targets in the next few iterations, and so predict their movement. Each agent, through its sensors, can know, from noisy measurements, which cell in its FoV is occupied and through those measurements the agents can formulate probabilistic beliefs about the expected positions of the targets in the next iteration. This is combined with a distributed process where the agents broadcast their current path and recent history of observations and movements which enables each agent to compute its own path that maximizes a global monitoring performance.

The beliefs are updated by each agent using a Bayesian filter, firstly predicting the next position of the target based on the current belief and then, taking into account the new observations, it updates the belief using Bayes' rule.

The method in [6] does not solve exactly the same problem as ours since this method is used for monitoring targets and not to locate them. However, target localization would be possible through the Bayesian filter, but it would take some extra steps in order to convert the belief into an actual estimate. The path planning method should also be changed because the optimization problem is not suitable for target localization.

In [7] an algorithm based on personality is proposed to solve the multi-robot observation of multiple moving targets. Again, since this is considered to be a general solution for tracking the targets, the agent can only produce measurement data if a target is inside its FoV at a given time.

That paper considers how should the control of the agents be computed in order to benefit the total coverage of the network instead of the coverage of each agent, so the cost function chosen for this problem is given by the coverage rate of the network. The method evaluates the target diversity via Shannon's entropy because it wants to observe the largest possible number of targets and at the same time wants all the targets to be detected and entropy increases with the number of different targets observed. It also introduces a parameter in the control that represents the trade-off between greedy and global optimization. This happens when, for example, a group of agents is covering the same target and another uncovered target is far away from the group. The agents do not want to leave the target that they are currently covering because for a period of time they will not cover any target, so their individual benefit will decrease despite the group benefit increases when an agent starts covering a new target.

In [8] we have an extension of what was proposed in [7]. In this paper, the same problem is considered and the same setup is used, but now another problem is added and it focuses on the formation of the network. The authors suggest the concept of flexible formation since many solutions consider rigid formations of agents to improve the performance of systems, but this type of formation also has problems associated with it.

The information present in [7] and [8] is different from other papers in the literature since it does not constitute a full solution for our problem, as they only propose simple modifications to an existing solution to compute the control of the network. Nevertheless, they show a different interpretation of the problem and provide a covering solution that can be used as a basis for simpler and more specific problems.

In [9] we return to the target tracking problem where a network of agents tries to track the state of a mobile target. As data model, the agents can perform partial-state noisy measurements that are given by a linear model. The method uses a combination of flocking and distributed Kalman-Consensus Filtering algorithms to both estimate the state of the target and to compute the control for the next positions of the network.

According to the authors, flocking is the behavior that a network of mobile agents can assume with the goal of maintaining the connections between the nodes and try to avoid collisions between them.

The path planning in this method is done in order to improve the collective information value for the network. The collective information is calculated from the measurement model and contains the amount of information in sensor data about the target position.

The Kalman-Consensus Filter is a distributed form of the Kalman Filter that performs consensus between the agents to fuse the computed information vector and matrix, so that each agent can then use the Kalman Filter to obtain not only an estimate of the current position of the target but also the control to move to its next position.

The use of the Kalman Filter implies that there are free parameters that have to be tuned before the algorithm can be successfully implemented. These make it harder to put this kind of algorithm into practice and to adapt it to new situations since the parameters have to be tuned again. On the other hand, the Kalman Filter has at its disposal much more data than a range-only solution, and, thus, can achieve better localization accuracy. The paper does not present the number of consensus iterations needed, so it is not possible to assess the communication payload of the method.

In [10], a similar solution to the previous one is presented. This paper considers a network of agents that move in a boundary of a determined space and a target that moves inside the defined space. The data model is composed by range measurements obtained by ultrasound-based sensors.

The target tracking process is handled by an Extended Kalman Filter performing a round of communications before the estimation to propagate the measurements from the agents through the network and then each node computes its own estimate.

Path planning along the boundary is made using the FIM to evaluate the amount of information that the network has about the target. The chosen optimization function was the determinant of the FIM, which they try to maximize.

The results from the paper only consider situations where the trajectory of the target is a predefined

eight-shaped curve. Adding this to the factor that the movement of the network is limited to the boundary of the space makes it very difficult to predict how this method would behave in different testing conditions. Also, considering that this method is based on an Extended Kalman Filter, it is possible to say that its implementation in different situations would be very difficult or even impossible due to the complexity and difficulty of implementation of this kind of method.

A complicating factor in the implementation is the lack of clear directions on how the distributed process should be used. It only unveils that there is a communication process between the agents and a fusing method for the data, but neither of them is presented.

The Extended Kalman Filter is a linearization of a nonlinear state estimation problem, highly dependent on the quality of the initialization. Unlike the linear Kalman Filter, if the initial estimate is bad it can diverge and provide unsatisfactory results.

In [11] we can see a specific case of a mobile robot equipped with sensors tracking a mobile target. The data model considered by the robot is the range measurement and the bearing when the target is inside the sector-shaped FoV. This is a centralized method since there is only one robot and it uses an Extended Kalman Filter to both locate the target and plan the path of the robot accordingly. This method showed an excellent performance in the presented tests against a state-of-the-art potential field approach. Nevertheless it only addressed the one-edge problem.

Some of the authors from the previous method also proposed a new method [12] to plan the state and control trajectories of a 2D network of sensors with an omnidirectional FoV that try to cooperatively detect a mobile target. As it is a detection method, it maximizes the detection probability, considering that if the target is inside the sensor's FoV it is detected. So, it considers some coarse localization through a detection framework. This is a centralized method for computing detection probabilities and an optimal control solution given both the maximization of the detection probability and the minimization of the control usage.

Since this method does not aim at improving localization precision it is not a full solution for our problem. Although this method could be combined with other target tracking methods, it is very difficult to implement, thus impeding massive deployment.

Now we will consider a variation from the general problem where there is no target. So, the network is formed by agents and anchors and the problem consists in self-locating the agents through communications between them and the anchors. This type of problem is considered in [13],[14] and [15].

The setup in [13] is a 2D network consisting in multiple agents and anchors. This paper emphasizes the process used to obtain the data model of the agents: they propose a new channel model that describes the relationship between the received signal strength statistics and the node locations. From the received signal strength statistics they are able to obtain noisy measurements that express the position of the agent relative to other agents and anchors.

The authors apply *Maximum Likelihood* (ML) estimation to self-localization in three different approaches, all of them assuming white Gaussian noise. The three approaches are consensus based, where the first considers the average consensus, where each agent computes the estimate based on the estimates from the neighboring agents, the second is the weighted average consensus approach

where the information from the other nodes of the network is weighted before the computations, and finally the weighted average consensus with accuracy exchange, where neighbor specific weights are added to speed up the convergence.

The numerical experiments in this paper comprises a network of static anchors and static agents where the three methods are tested and their performance is evaluated. Different network configurations are tested from low connectivity between agents and anchors to all-to-all connectivity where only one consensus round is required to merge information from all the nodes of the network. The weighted average consensus with accuracy exchange method showed the best performance for the different configurations, getting close to the performance of the centralized method but it is never able to reach that kind of performance since the weighted average consensus with accuracy exchange method does not converge exactly to the centralized estimate.

The use of a distributed consensus-based maximum likelihood estimator makes this method the one that provides the most similar solution to the one presented in this thesis, in terms of distributed estimation. However, the algorithm uses a different data model and only considers static networks, so it is not possible to see how well it will handle mobile networks. Finally, the authors only compare their distributed methods with the centralized estimator, which makes it difficult to draw conclusions on how it would perform when compared with different state-of-the-art localization methods.

In [14] we can see the use of a Kalman Filter to estimate the positions of mobile agents from fixed anchors using as data model range measurements computed from time of arrival measurements. This data model induces a non-linear optimization function. To be able to use the standard Kalman Filter, the authors try to apply the known locations of the anchors, obtaining a set of linear equations in the unknown locations. Now they have a linear model and so it is possible to localize the moving nodes in an adaptive manner using the Kalman Filter. This is also a simpler method because it is centralized.

This centralized Kalman Filter approach produces better results than the distributed one for the network it is configured for. However, the centralized architecture decreases the flexibility of this kind of filter because now we also have to consider the problems for this type of architecture already presented in Section 1.2 in addition to the usual free parameters that need to be tuned.

In [15] a variant of this problem is explored. Agents are deployed to map the space where the network lies. In this method, the agents take measurements using a Laser Range Finder ray. The measurements are fed to a Kalman Filter to estimate the position of the agents and select the next action to be performed. The evaluation of the actions of the agent is based on the localizability matrix, which is the result of a discretization of the FIM based on a probabilistic grid map. From the inverse of the eigenvalues of the localizability matrix it is possible to obtain the localization probability distribution ellipses which are used to evaluate the positions of the agents, and the movement is then made in order to get the lowest expected localization covariance. After selecting an action with the lowest expected localization covariance the other agents should select complementary actions in order to eliminate maximum localization uncertainties. So this method considers cooperative localization, but in a centralized architecture.

Reference [16] considers the 1D setup with one sensor and one target. The data model is assumed to be known and defined as a function of both the deterministic sensor position and the unknown target

position, to which zero-mean noise is added. The problem consists in a sensor that moves along one direction and tries to locate a moving target that moves along a parallel direction. The Fisher Information Matrix is again used for path planning and it is computed from the likelihood. The expected information density matrix comes from the expectation of the Fisher Information over the belief. The expected information density measures the quantity of information that the sensor has about a certain area and from it it is possible to create a control for the sensor. The best control is the one that makes the sensor often move outside of regions of maximum expected information in order to maintain good estimates for the object positions. A Bayesian update is also used to update the belief and do the target tracking. Since there is only one sensor this method is centralized.

In [17] an update to the previous method is presented, including tests with real equipment. Now the interest region is in the 2D case and the authors also contemplate multiple mobile targets. The sensor measures disturbances in a self-generated weak electric field and uses these measurements to update the belief of the target positions using a Bayesian filter. Again, from the Fisher Information Matrix it is possible to compute the expected information matrix. Control of the sensor is based on ergodicity and is made so that the path followed should focus on regions with higher expected information, since this is where useful measurements are more likely. This is made through minimization of a cost function that includes two terms, one being the ergodic cost and the other the control effort.

Since this method only considers a single sensor is hard to think how would it perform if it was implemented in a network of sensors. For the network of sensors case the control would also need to be more complex since now we do not need the whole network to focus on a single high expected information density area and some of the sensors can explore other regions covering more space.

In Section 1.1 it was already mentioned that this type of problem is heavily present in the autonomous vehicles area meaning that a lot of solutions are also present in the literature, as seen in [18],[19],[20] and [21]. Although addressing a problem with this level of specificity, sometimes these solutions can be adapted to also solve the more general problem.

In [18] the control strategies for a team of aerial vehicles performing distributed and cooperative sensing are considered. To compute those control strategies a 3D setup of aerial vehicles equipped with a vision-based system, composed by cameras, use the data obtained by the images to compute likelihood functions that indicate the probability of the given data considering the real target position, used as the data model.

After obtaining the likelihood functions, a recursive Bayesian framework is used to form a probability distribution of the target position, assuming that the likelihood function computed by each vehicle is passed through the network, so every vehicle is able to compute the distribution for the target position.

In the path planning phase, the goal is to maximize a reward function that represents the value of the information collected from all vehicles and using a gradient-based approach it is possible to find the optimal path for each vehicle. Long-term planning is also considered, in which the vehicle only plans a path if the measurements made at a certain time have a value above some defined threshold. If this does not happen the vehicle continues to follow the previously planned path.

The used data model may not be the best to solve the more general problem at hand since it brings

more error to the estimation process. However, it is understandable that for the specific case considered a data model like the one we considered may not be viable. Another aspect that has to be considered in this case is the maneuverability of the aerial vehicles that may be affected by fast changing paths and moving targets, which is not a problem when considering the general case since it does not need to have a pre-planned path because it can at every iteration choose just the next position of each agent of the network which will form a path.

The work in [19] examines a setup of a team of multiple aerial vehicles and one target, in which the vehicles are equipped with 3D ranging sensors. The data model is the range measurements that each vehicle with known position computes to the target. This method considers two types of control that are handled in different ways, both considering the existence of, in the authors' words, a sweet spot located at a certain distance from the target and where the uncertainty in the measurements is minimum. This sweet spot is the position in the space that the vehicle tries to reach using the computed control.

For cooperative control a Kalman-Bucy filter is used which compute an estimate of the target position and the control for the vehicles based on global position-error covariance. This covariance matrix is evaluated under three different criteria to build three cost functions from which the control is computed in order to show the generality of the approach. The criteria are minimizing the uncertainty ellipsoid for the estimates, minimizing the length of the largest axis of the same ellipsoid and suppressing the average variance of the estimates. The authors claim that their paper is not specifically concerned with the problem of designing distributed control laws, so the method presented does not specifically state how the cooperation occurs, but they also point out that some distributed strategies can be adapted to fit their method.

For non cooperative control the agents compete with each other to find the best estimate of the target position. The estimates are produced by coupled Kalman filter-like equations.

In the previously proposed solution the vehicles had full knowledge about their positions and orientations but the paper also proposes a method to address the same problem when the vehicles do not know those measurements exactly. This makes the vehicles perform measurements between them in addition to the ones relative to the target, and compute the control such that the resultant trajectories minimize the uncertainty of the position of the other vehicles as well as the target.

This second problem is not aligned with the topic of this thesis since we assume that the agents of the network have full knowledge of their positions and so there is no need to increase the complexity of the problem in that direction.

The work in [20] deals again with the setup of multiple aerial vehicles and one target. The proposed method starts by using a general data model from which a recursive Bayesian filter can be used to estimate the position of the target. Cooperative estimation is also used where the measurements are passed through the network of vehicles, but this paper does not focus on how the data is shared. The Sequential Monte-Carlo method is then applied to approximate the posterior obtained by the Bayesian filter using samples and weights. After the use of the Monte-Carlo method, instead of having the probability distributions for the target position, it considers a set of weighted particles where each particle is a possible solution for the target position according to the probability distribution given by the previously

computed posterior.

For the path planning phase a specific data model is now considered consisting in the received signal strength indicator with omnidirectional antenna where considering the level of the *Signal to Noise Ratio* (SNR) it is possible to find the best detection point for the aerial vehicle if the SNR is more than a threshold. The addressed problem is the maximization of the probability of target detection. The guidance law used to guide the vehicle to the best detection point is an acceleration vector computed from the current state of the vehicle, the desired state of the vehicle that corresponds to the best detection point and other quantities that are needed in the movement of aerial vehicles.

The simulation results for this method show that, for the setup of only one vehicle and one mobile target, when signal occlusions occur (i.e., the line of sight is blocked by an object) the vehicle will normally record a lower signal than the previous positions so it will loiter until a higher signal is received. This is a common problem when dealing with signals and objects in the line of sight of the agents, but can be handled by a distributed network of agents since the agents that records worse or no information about the target can use the information given by the rest of the network to make the computations in the same way as if it received a good signal. This subject is also addressed in the results of this paper when a network of multiple aerial vehicles is considered. A similar problem that was not consider in this paper is how to deal with signal occlusion when the the signal considered is used to perform the communications between vehicles.

In [21] an even more specific problem is presented. It consists in the centralized path design of only two aerial vehicles considering one stationary emitter, using as data model time-difference-of-arrival measurements from which range difference measurements are easily obtained.

The cost function used for path planning is computed from the FIM, which is obtained from the unit vectors pointing from the emitter to the sensors. From the FIM and considering that the goal of this method is to find the optimal trajectories that minimize the location error considering a set of constraints, the authors choose to maximize the minimum eigenvalue of the FIM as the cost function, which is the same as maximizing the quantity of information that the network has about the emitter and also the same as minimizing the localization error. This is also a special case because various constraints are also considered in this problem. The applied constraints are related to restrictions that the vehicles are affected by in the practical application of the proposed method like no fly zones or speed and maneuvering limitations.

To solve this type of problem the authors suggest a method based on the augmented Lagrangian method known as the alternating direction method of multipliers. The augmented Lagrangian method or method of multipliers is used to solve constrained optimization problems, where these are considered as a series of unconstrained problems with another term added to the objective, mimicking a Lagrange multiplier, replacing the regularization with a quadratic penalty term on the constraints used in other penalty methods. The alternating direction method of multipliers follows the augmented Lagrangian method but introduces new variables to break the problem into smaller pieces and solves the optimization problem alternating between variables considering that the others are fixed. This leads to an iterative method to compute the control for the aerial vehicles and to produce an estimate for the target position. The

control is updated via semi-definite relaxation and the estimate for the target position is updated via majorization-minimization.

As one can see the introduction of constraints in an optimization problem usually makes it more difficult to solve, and this paper presents a way to handle these constraints that are usual in this type of problems that are this close to the practical application of the method. In a more generalized case these specific constraints are not considered, so there is no need for such complex method to solve the problem.

As seen in the presented literature review, many methods exist to perform the target localization and path planning of the sensors and so also many different combinations of the two can be found. A lot of methods use, like our solution, the minimization of the estimation error or maximization of the information as cost to perform the path planning, being this method the one that is more repeated through the presented literature review.

For target tracking the two mainstream approaches are based on the Bayesian framework or the Kalman filter. Both methods are used as a basis for more advanced and detailed methods applied to also more specific problems. These have advantages and disadvantages that are stated across the review. Our solution resorts to an alternative method that is less frequently seen in the literature.

Finally, it is also important to say that many of these papers claim that their methods are distributed but do not discuss in detail how the sharing of information works which, is a big disadvantage when trying to reproduce the same results. We hope that our solution also helps to address this frustrating issue.

2.2 Benchmark

Now we will introduce the benchmark that will be used in Section 5.4 to compare with our solution. This will provide us with numerical evidence that can justify how well our solution preforms when solving the problem compared to a published method.

For the benchmark we wanted a method that would share some of the characteristics of our solution. So we looked for a method that was also distributed, used range measurements as its data model and that could do target tracking and path planning as well. It was important to find a method that was also distributed because the centralized architecture considers a lot more information in its computations than the distributed counterpart, meaning that it will provide better results. The distributed architecture is a partitioning of the centralized one and converges to it, so in the best scenario, the distributed architecture will (asymptotically) provide the same results, although having advantages mainly in flexibility as presented in Chapter 1.

The benchmark's data model has to be similar to the one we consider in our solution because the agents have to work with the same measurements in order to produce the same results. If we are comparing methods with different data models they will work with different types of measurements that will also include different types or quantities of noise. This means that the results will be different and cannot be compared. So we have to choose a benchmark that also uses the range measurements as

its data model.

Finally, we chose a method that includes both target tracking and path planning because we wanted a complete method to which we can fully compare our solution without having to combine a method for target tracking from one author and a method for path planning from another.

Having all these features in mind we chose the method presented in [3]. It comprises a distributed cooperative method for Bayesian estimation and control consisting in two layers: the estimation layer where it performs target tracking feeding the control layer generating the control of the agents. But we cannot use this method as it is presented since it considers a network composed by static anchors that have full knowledge of their positions, cooperative agents and targets. The main objective of this method is cooperative localization of the agents with measurements from the anchors and from other neighboring agents and then, after having knowledge of their positions, the agents estimate the position of the target. In our solution we assume that the sensors have full knowledge of their current positions so we have to consider a special case also presented in [3] where only the estimation of the target position is considered.

The main goal of the estimation layer is to estimate the position of the target using prior information and all past and present range measurements, as defined in (1.1). To do so the authors of this method resort to Bayesian statistics.

Bayesian statistics [22],[23] is a field of statistics in which the unknown quantities are treated as random variables with known distributions instead of simply unknown deterministic quantities. Basically, in this Bayesian approach we assume that we know the *probability density function* (pdf) of the distribution that models the considered random variable and from measurements relative to that variable we can derive a new pdf. In Bayesian terms we say that by postulating the prior probability distribution we try to find the posterior probability distribution that considers all the information that the measurements have about that random variable. The posterior is derived from the prior using Bayes' rule and the likelihood, which is a conditional distribution that shows the compatibility between the measurements and its parameters. Bayes' rule also includes the use of the marginal likelihood, which is simply the distribution of the measurements, and since the marginal likelihood does not vary with estimators one can simply ignore it and compute the relative probabilities instead.

Going back to the method under study, the Bayesian statistics are used to estimate the marginal posterior probability distribution that models the target position. This marginal posteriors only take into account the posterior for the last iteration of the algorithm and are marginals of the joint posteriors that encompass all the posterior and measurement data from all iterations. The marginal posteriors are an update from the priors using the Bayes' rule with the range measurements obtained in that iteration. Since we are only trying to estimate the position of the target its marginal posterior is proportional to its prior and all the likelihoods corresponding to the measurements from all the agents.

To compute this marginal posterior a distributed, cooperative algorithm that combines *Belief propagation* (BP) message passing and the average consensus scheme is implemented. This algorithm then provides an approximation of the marginal posterior which they call a belief. The belief is an approximation of the posterior since it needs the measurements of all the agents to be computed and since only

the agent has its own measurements they have to exchange this data and so each agent only can can compute an approximation to the centralized computations. The concept of consensus will be explained in more detail in Section 3.3. As a result of these computations, each agent has its own belief for the position of the target. Finally, it is necessary to feed the control layer with a common belief which can be achieved with a belief consensus algorithm. Now every agent has the same belief for the position of the target. Each belief is represented by a defined number of samples that are obtained according to the resulting pdf. The estimate for the target position can be computed through the *Minimum Mean Square Error* (MMSE) estimator.

The MMSE estimator [23] is the estimator that minimizes the Bayesian mean square error and can be seen as the Bayesian approach of the MVU estimator. The MMSE estimator consists on obtaining an estimate for the quantity that we are working with from the posterior and it is simply given by the mean of the posterior PDF.

Again returning to the benchmark, the MMSE estimate is simply obtained by the arithmetic mean of the values for all the samples that represent the obtained belief. To compute a sample-based approximation for sequential state estimation in this problem, the authors use a particle filter [24].

The control layer is where the next positions of the agents are estimated through an information-seeking controller. This controller moves the agents in order to maximize the negative joint posterior entropy of the target at the next time step, conditioned on all measurements in the network, also in the next time step.

The entropy [23] is used as a measurement of the randomness of a random variable. Maximizing the information about a certain random variable entails minimizing its randomness, whichs is why an information-seeking controller chooses to maximize the negative joint entropy. The joint entropy is computed from the posteriors computed in the estimation layer and the joint distribution of the target position and the range measurements. The computation of the control used to move the agents is made through a gradient ascent [25] using the joint posterior entropy as the cost function in order to maximize it. The authors claim that the gradient is computed by each agent using a distributed consensus-based or flooding-based method. The flooding method is different from the consensus one in the process of sharing the information from each agent through the network. In the flooding method each agent broadcasts its information to its neighbors and the neighbors will broadcast that same information to the other agents. This way all the agents have the information from the other agents at their disposal before starting the computations. In the consensus process the information that one agent broadcasts to its neighbors will be averaged to make computations by them and just the result of that computations will be broadcast to their neighbors, so the the agents do not have direct access to all the original information, but rather to a more "dilluted" form. In the flooding method each agent ends up solving the centralized problem because more information is shared between the agents, but this also increases the time and energy that is consumed through the process, so this method is only effective in small networks.

From the use cases in [3] we only have interest in the cooperative self-localization and target tracking sections. Here the authors mention our exact application and refer how to adapt to their method. Results for the tracking of a mobile target are presented in the form of the *Root-Mean-Square Error* (RMSE),

at each time step, which expresses the error between the computed estimate for the target position and the real position. From the results we can infer that the method achieves low RMSE values for the performed experiments in a low number of iterations if we assume that the first iterations are used for self localization.

When comparing our solution with the benchmark method it is already possible to see that the benchmark uses different processes to estimate the target position and to control the agents. It also entails more information exchange rounds which propagate more information through the network when compared to our solution that works with less information. On the other hand, this increase in information traffic also increases the communications cost that can be prohibitive in some deployments and so one may want to minimize this cost as well.

Chapter 3

Proposed Method

The mathematical approach to the problem proposed in Chapter 1 will be presented and explained in this chapter.

In order to give an easier and clearer mathematical explanation of our approach, a setup consisting of only one target and multiple agents will be considered in this chapter. A setup with multiple targets will be analyzed in Chapter 4.

3.1 Linear Model

Following the explanations given about the SR-LS in Section 1.3 and as considered in [2] we seek to find a mathematical solution to this least-squares problem. So we start by writing the SR-LS cost function as

$$\min_{\mathbf{p}(t)\in\mathbb{R}^d} \sum_{i=1}^{n} (||\mathbf{s}_i(t) - \mathbf{p}(t)||^2 - \mathbf{r}_i^2(t))^2. \tag{3.1}$$

The goal is to find the position of the target $\mathbf{p}(t)$ which minimizes this cost function. As it was said before, this problem is nonconvex and so a solution for it is very difficult or even impossible to find. The only option is trying to find a similar convex optimization problem that can provide an accurate approximation to the desired solution of the SR-LS problem.

To design this convex optimization problem we will use the definition of range found in Section 1.5. To simplify the math involved for now, we will consider that the noise $\mathbf{w}$ is so small that it can be neglected. So we start with an adaptation of (1.1) and calculate the squared range

$$r_i(t) = ||\mathbf{s}_i(t) - \mathbf{p}(t)|| \Leftrightarrow$$
$$r_i^2(t) = ||\mathbf{s}_i(t) - \mathbf{p}(t)||^2 \Leftrightarrow$$

$$r_i^2(t) = ||\mathbf{s}_i(t)||^2 + ||\mathbf{p}(t)||^2 - 2\mathbf{s}_i(t)^T\mathbf{p}(t). \tag{3.2}$$

After some algebraic manipulations on equation (3.2) we obtain

$$r_i^2(t) - ||\mathbf{s}_i(t)||^2 = ||\mathbf{p}(t)||^2 - 2\mathbf{s}_i(t)^T\mathbf{p}(t) \tag{3.3}$$

and considering the set of range measurements of all agents, we can formulate it as

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{3.4}$$

where

$$\mathbf{y} = \begin{bmatrix} r_1^2(t) - ||\mathbf{s}(t)_1||^2 \\ \vdots \\ r_n^2(t) - ||\mathbf{s}(t)_n||^2 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 1 & -2\mathbf{s}_1(t)^T \\ \vdots & \vdots \\ 1 & -2\mathbf{s}_n(t)^T \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} ||\mathbf{p}(t)||^2 \\ \mathbf{p}(t) \end{bmatrix}. \tag{3.5}$$

To clarify the mathematical expressions presented in the rest of the chapter we will not write the time dependence for some variables. Now re-introducing the noise that was neglected we obtain

$$\mathbf{y} = \mathbf{A}\mathbf{x} + 2\Xi\mathbf{w} + (\mathbf{w} \circ \mathbf{w}) \tag{3.6}$$

which represents the linear model of the problem under consideration and where $(\mathbf{w}\circ\mathbf{w})$ is the Hadamard product between the same noise vector given by

$$(\mathbf{w} \circ \mathbf{w}) = \begin{bmatrix} w_1^2 \\ \vdots \\ w_n^2 \end{bmatrix} \tag{3.7}$$

and

$$\Xi = \begin{bmatrix} ||\mathbf{s}_1 - \mathbf{p}|| & & \\ & \ddots & 0 \\ 0 & & ||\mathbf{s}_n - \mathbf{p}|| \end{bmatrix}. \tag{3.8}$$

By analyzing the model of the problem it was found that we have to consider two components of noise that influence the range measurements for each agent. The first one comes from the electronic part of the agent and is considered to be equal for all the agents. This default noise will be represented by $\hat{\sigma}$. The other component that influences the measurements is the distance of the agent relative to the target. This means that an agent that is far away from the target will have a less accurate range measurement relative to it in comparison to an agent which is close to that same target. Since

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{3.9}$$

and $w_i \sim \mathcal{N}(0, \sigma_i^2)$, which represents the noise in the range measurement of agent i at time t, we can introduce the two components described before defining the standard deviation of each agent as $\sigma_i = \beta||\mathbf{s}_i - \mathbf{p}||\hat{\sigma}$, where β is a scalar variable that dictates the influence of the distance on the noise and $\beta\hat{\sigma}$ is considered a dimensionless quantity.

The linear model obtained still does not turn this problem into a convex one. As seen in [2] and [26] one possible way to make this problem convex is to neglect the dependence of $x_1 = ||\mathbf{p}||$ on the other elements of the solution vector $\mathbf{x}$. So, considering the mentioned approach, we have a convex problem and there are multiple methods to solve it. We will try to find an estimator $\hat{\mathbf{x}}$ that makes the equation (3.6) true. This is the same as solving the *Unconstrained Squared-Range-Based Least Squares* (USR-LS) problem given by

$$\min_{\mathbf{x} \in \mathbb{R}^{d+1}} ||\mathbf{A}\mathbf{x} - \mathbf{y}||^2. \tag{3.10}$$

3.2 Linear Estimators

Now we will present a method to solve the problem proposed in Section 3.1. Since we already reached a linear form for the problem considered we chose to use the *Minimum-Variance Unbiased* (MVU) linear estimator.

The MVU estimator is the unbiased estimator that has lower variance than any other. This estimator has the advantage that it is easy to obtain once our problem is described by a linear model and other useful statistical quantities can be easily obtained as well.

According to [23], start with a simple linear model formulated as

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w} \tag{3.11}$$

where $\mathbf{w}$ is white Gaussian noise where $\mathbf{w} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$, and $\mathbf{x}$ is the vector containing the parameters that are going to be estimated.

It is known that the *Cramer-Rao Lower Bound* (CRLB) expresses a lower bound on the variance of an unbiased estimator. An unbiased estimator that attains the CRLB for all $\mathbf{x}$ must satisfy

$$\frac{\partial \log(p(\mathbf{y}; \mathbf{x}))}{\partial \mathbf{x}} = \mathbf{I}(\mathbf{x})(\mathbf{g}(\mathbf{y}) - \mathbf{x}) \tag{3.12}$$

where $\hat{\mathbf{x}} = \mathbf{g}(\mathbf{y})$ is the MVU linear estimator, and $\mathbf{I}^{-1}(\mathbf{x})$ is the covariance matrix of $\hat{\mathbf{x}}$. The covariance matrix is a matrix where each element represents the covariance between two random variables. The covariance is a measure of joint variability between those variables. This means that using the covariance it is possible to measure the dependence between the two variables. It is also important to refer that $\mathbf{I}(\mathbf{x})$ is the FIM that will be studied in more detail in Section 3.4.

After some substitutions and algebraic manipulations we obtain

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \tag{3.13}$$

$$\mathbf{I}(\mathbf{x}) = \frac{\mathbf{A}^T\mathbf{A}}{\sigma^2} \tag{3.14}$$

and

$$\mathbf{C}_{\hat{\mathbf{x}}} = \mathbf{I}^{-1}(\mathbf{x}) = \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}. \tag{3.15}$$

Since the estimator is just a linear transformation of a normally distributed vector $\mathbf{y}$, the statistical performance of $\hat{\mathbf{x}}$ is completely specified and given by

$$\hat{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}). \tag{3.16}$$

Going back to our linear model presented in (3.6), we see that the noise term presented is not simple white Gaussian noise. Fortunately, [23] also considers the problem where the noise is not white. This general linear model assumes $\mathbf{w} \sim \mathcal{N}(0, \mathbf{C})$. The MVU linear estimator and its covariance matrix for the general linear model are

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{C}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C}^{-1}\mathbf{y} \tag{3.17}$$

$$\mathbf{C}_{\hat{\mathbf{x}}} = (\mathbf{A}^T\mathbf{C}^{-1}\mathbf{A})^{-1}. \tag{3.18}$$

In (3.6) it is also possible to see that the noise term given by $(\mathbf{w} \circ \mathbf{w})$ is not Gaussian at all, meaning that its expected value is not zero, influencing the problem as a bias. Since there is a bias in our problem the MVU linear estimator can not be implemented in a straightforward way. Two methods were developed in order to solve the bias problem.

3.2.1 Method 1

The first to solution is to see that the term $(\mathbf{w} \circ \mathbf{w})$ considers that the noise term in the measurements is squared and considering that the noise is a small quantity this term will be much smaller than the other noise term that is multiplied by the distance between the agents and targets. So we neglect the squared noise term and obtain the following linear model

$$\mathbf{y} = \mathbf{A}\mathbf{x} + 2\Xi\mathbf{w}. \tag{3.19}$$

Now it is possible to apply the MVU linear estimator to solve the problem presented in (3.19) but we still have to calculate $\mathbf{C}$ which corresponds to $\mathbf{C}_{\mathbf{y}}$ that is the covariance matrix of the range measurements. To calculate this matrix we will use the basic definition of covariance for element ij given by

$$C_{\mathbf{y}_{ij}} = E\big[(y_i - E[y_i])(y_j - E[y_j])\big] \tag{3.20}$$

where $E[X]$ represents the expected value of random variable X. Taking the i-th line from (3.19) we get

$$y_i = ||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i - ||\mathbf{s}_i||^2, \tag{3.21}$$

replacing this into equation (3.20) we get

$$\begin{aligned} C_{\mathbf{y}_{ij}} = E\big[&(||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i - ||\mathbf{s}_i||^2 - E[||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i - ||\mathbf{s}_i||^2]) \\ &(||\mathbf{s}_j - \mathbf{p}||^2 + 2||\mathbf{s}_j - \mathbf{p}||w_j - ||\mathbf{s}_j||^2 - E[||\mathbf{s}_j - \mathbf{p}||^2 + 2||\mathbf{s}_j - \mathbf{p}||w_j - ||\mathbf{s}_j||^2])\big]. \end{aligned} \tag{3.22}$$

Since the expected values of the norms are the same as the norms, recalling $w_i(t) \sim \mathcal{N}(0, \sigma_i^2)$ we have $E[w_i] = 0$ and after doing the multiplication between the two terms we have

$$C_{\mathbf{y}_{ij}} = E\big[4||\mathbf{s}_i - \mathbf{p}|| \cdot ||\mathbf{s}_j - \mathbf{p}||w_i w_j\big]. \tag{3.23}$$

To finalize the demonstration we have to calculate specifically the two different cases of elements inside the matrix. These elements are the ones forming the diagonal and the others. So considering that $i = j$ and that $E[w_i^2] = \sigma_i^2$ we can compute the diagonal of the covariance matrix obtaining

$$C_{\mathbf{y}_{ii}} = 4||\mathbf{s}_i - \mathbf{p}||^2\sigma_i^2. \tag{3.24}$$

The remaining positions of the matrix are obtained doing the same procedure as before but with $i \neq j$. Since the noise terms of i and j are independent from each other $E[w_i w_j] = 0$ and so

$$C_{\mathbf{y}_{ij}} = 0 \tag{3.25}$$

which was expected since the range measurements of each agent relative to the target are independent to the measurements of the other agents of the network. Therefore, matrix $C_{\mathbf{y}}$ is given by

$$\mathbf{C_y} = \begin{bmatrix} 4||\mathbf{s}_1 - \mathbf{p}||^2\sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & 4||\mathbf{s}_n - \mathbf{p}||^2\sigma_n^2 \end{bmatrix} \tag{3.26}$$

and since this is a diagonal matrix it is easy to get its inverse

$$\mathbf{C_y}^{-1} = \begin{bmatrix} \frac{1}{4||\mathbf{s}_1 - \mathbf{p}||^2\sigma_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{4||\mathbf{s}_n - \mathbf{p}||^2\sigma_n^2} \end{bmatrix}. \tag{3.27}$$

Remember that σ_i was defined in the previous section as $\sigma_i = \beta||\mathbf{s}_i - \mathbf{p}||\hat{\sigma}$.

So the MVU linear estimator and covariance matrix for the linear model defined in (3.19) are respectively given by

$$\hat{\mathbf{x}} = (\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{y} \tag{3.28}$$

$$\mathbf{C}_{\hat{\mathbf{x}}} = (\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{A})^{-1} \tag{3.29}$$

and so

$$\hat{\mathbf{x}} \sim \mathcal{N}(\mathbf{x}, \mathbf{C}_{\hat{\mathbf{x}}}), \tag{3.30}$$

where $\hat{\mathbf{x}}$ is a $(d+1) \times 1$ vector and $\mathbf{C}_{\hat{\mathbf{x}}}$ is a $(d+1) \times (d+1)$ matrix. For example, when working in $\mathbb{R}^2$ their sizes become 3×1 and 3×3 respectively.

3.2.2 Method 2

The alternative approach is to consider the linear model presented in (3.6) and use a linear estimator to solve it. However the linear estimator is biased and so it does not guarantee that minimum variance is achievable. The covariance matrix $\mathbf{C}_{\mathbf{y}}$ for this kind of estimator will be slightly different from the one presented in (3.38). In this case we start by considering the i-th line from (3.6) given by

$$\begin{aligned} y_i &= (||\mathbf{s}_i - \mathbf{p}|| + w_i)^2 - ||\mathbf{s}_i||^2 \Leftrightarrow \\ y_i &= ||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i + w_i^2 - ||\mathbf{s}_i||^2, \end{aligned} \tag{3.31}$$

replacing this into equation (3.20) we get

$$\begin{aligned} C_{\mathbf{y}_{ij}} = E\big[&(||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i + w_i^2 - ||\mathbf{s}_i||^2 - E[||\mathbf{s}_i - \mathbf{p}||^2 + 2||\mathbf{s}_i - \mathbf{p}||w_i + w_i^2 - ||\mathbf{s}_i||^2]) \\ &(||\mathbf{s}_j - \mathbf{p}||^2 + 2||\mathbf{s}_j - \mathbf{p}||w_j + w_j^2 - ||\mathbf{s}_j||^2 - E[||\mathbf{s}_j - \mathbf{p}||^2 + 2||\mathbf{s}_j - \mathbf{p}||w_j + w_j^2 - ||\mathbf{s}_j||^2])\big]. \end{aligned} \tag{3.32}$$

Again, the expected values of the norms are the same as the norms and $E[w_i] = 0$. After doing the multiplication between the two terms we have

$$\begin{aligned} C_{\mathbf{y}_{ij}} = E\big[&4||\mathbf{s}_i - \mathbf{p}|| \cdot ||\mathbf{s}_j - \mathbf{p}||w_i w_j + 2||\mathbf{s}_i - \mathbf{p}||w_i w_j^2 - 2||\mathbf{s}_i - \mathbf{p}||w_i E[w_j^2] + 2||\mathbf{s}_j - \mathbf{p}||w_i^2 w_j + w_i^2 w_j^2 \\ &- w_i^2 E[w_j^2] - 2||\mathbf{s}_j - \mathbf{p}||w_j E[w_i^2] - w_j^2 E[w_i^2] + E[w_i^2]E[w_j^2]\big] \end{aligned} \tag{3.33}$$

Since w is normally distributed we can use the moments [27] of this variable to know the value of $E[w^m]$. Since w has zero mean, this means that this random variable is centered, we can use the central moments given by

$$\begin{aligned} E[w^2] &= \sigma_i^2 \\ E[w^3] &= 0 \\ E[w^4] &= 3\sigma_i^4, \end{aligned} \tag{3.34}$$

For the diagonal of the matrix we know that $i = j$ and so we have

$$C_{\mathbf{y}_{ii}} = 4||\mathbf{s}_i - \mathbf{p}||^2 E[w_i^2] + 2||\mathbf{s}_i - \mathbf{p}||E[w_i^3] - 2||\mathbf{s}_i - \mathbf{p}||E[w_i]E[w_i^2] + 2||\mathbf{s}_i - \mathbf{p}||E[w_i^2]E[w_i] + E[w_i^4]$$
$$-E[w_i^2]E[w_i^2] - 2||\mathbf{s}_i - \mathbf{p}||E[w_i]E[w_i^2] - E[w_i]^2 E[w_i^2] + E[w_i^2]E[w_i^2], \tag{3.35}$$

and applying the moments presented in (3.34) we get

$$C_{\mathbf{y}_{ii}} = 4||\mathbf{s}_i - \mathbf{p}||^2 \sigma_i^2 + 2\sigma_i^4. \tag{3.36}$$

For the remaining elements of the matrix where $i \neq j$ the central moments simplify the expression until we get

$$C_{\mathbf{y}_{ij}} = 0. \tag{3.37}$$

Therefore, matrix $\mathbf{C_y}$ is given by

$$\mathbf{C_y} = \begin{bmatrix} 4||\mathbf{s}_1 - \mathbf{p}||^2 \sigma_1^2 + 2\sigma_1^4 & & \\ & \ddots & 0 \\ 0 & & 4||\mathbf{s}_n - \mathbf{p}||^2 \sigma_n^2 + 2\sigma_n^4 \end{bmatrix} \tag{3.38}$$

and its inverse

$$\mathbf{C_y}^{-1} = \begin{bmatrix} \frac{1}{4||\mathbf{s}_1 - \mathbf{p}||^2 \sigma_1^2 + 2\sigma_1^4} & & \\ & \ddots & 0 \\ 0 & & \frac{1}{4||\mathbf{s}_n - \mathbf{p}||^2 \sigma_n^2 + 2\sigma_n^4} \end{bmatrix} \tag{3.39}$$

The linear estimator and covariance matrix for the linear model defined in (3.6) are respectively given by

$$\hat{\mathbf{x}} = (\mathbf{A}^T \mathbf{C_y}^{-1} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{C_y}^{-1} \mathbf{y} \tag{3.40}$$

$$\mathbf{C}_{\hat{\mathbf{x}}} = (\mathbf{A}^T \mathbf{C_y}^{-1} \mathbf{A})^{-1} \tag{3.41}$$

and the bias of the estimator is obtained by computing $E[\hat{\mathbf{x}}] - \mathbf{x}$ which gives

$$E[\hat{\mathbf{x}}] - \mathbf{x} = (\mathbf{A}^T \mathbf{C_y}^{-1} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{C_y}^{-1} E[\mathbf{y}] - \mathbf{x} \tag{3.42}$$

since the $E[\mathbf{w}] = 0$ we have that

$$E[\mathbf{y}] = \mathbf{A}\mathbf{x} + E[(\mathbf{w} \circ \mathbf{w})] \tag{3.43}$$

and applying this in equation (3.42) we get

$$E[\hat{\mathbf{x}}] - \mathbf{x} = (\mathbf{A}^T\mathbf{C}_\mathbf{y}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C}_\mathbf{y}^{-1}\boldsymbol{\Psi}, \tag{3.44}$$

where

$$\boldsymbol{\Psi} = \begin{bmatrix} \sigma_1^2 \\ \vdots \\ \sigma_n^2 \end{bmatrix}. \tag{3.45}$$

The estimator is given by a distribution which results from a mixture between the Gaussian distribution from the first noise term and a chi-squared distribution from the second, centered in the unbiased solution plus the bias, with mean $\mathbf{x} + (\mathbf{A}^T\mathbf{C}_\mathbf{y}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C}_\mathbf{y}^{-1}\boldsymbol{\Psi}$ and variance $\mathbf{C}_{\hat{\mathbf{x}}}$.

So it is possible to conclude that the estimates will always be biased but if the bias is small enough they can approximately be centered in the true solution.

Since the method presented in Section 3.2.1 uses a well known estimator and is much simpler, the converge proofs that are needed in Section 3.3 are much easier and so we chose it to be the main method considered in this work.

3.3 Distributed Linear Estimator

The MVU linear estimator presented in Section 3.2.1 is a centralized method since the estimator is computed with full knowledge of the measurements and positions of all agents, contained in $\mathbf{y}$ and $\mathbf{A}$, respectively.

In Chapter 1 it was said that one of the major properties of this method is distributed computation, so a new distributed approximation to the MVU linear estimator, to which we call ISEE.U estimator, is presented in this section. With this method, each agent will be able to compute an approximation to the MVU linear estimator and the covariance matrix for the estimator by itself using only data from his neighbors. The distributed agreement on the estimate will be enabled by a consensus-like method.

The consensus consists in obtaining some specific value from a network on which all nodes agree. For this to happen it is necessary for a node to compute the value using only the measurements that it can obtain and then transmit its own value to the other nodes and receive their values in return. After the communications between nodes a new value will be computed in each node but now it will consider data that came from other nodes. After some iterations it is possible to reach a value that is agreed upon by the entire network.

In our case we start by computing the centralized version of the inverse covariance matrix $\mathbf{C}_{\hat{\mathbf{x}}}^{-1}$ given by $\mathbf{A}^T\mathbf{C}_\mathbf{y}^{-1}\mathbf{A}$. From (3.5) we have $\mathbf{A}$ and we can get $\mathbf{C}_\mathbf{y}^{-1}$ from (3.27). We can replace both matrices in the expression and get

$$\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{A} = \begin{bmatrix} 1 & \cdots & 1 \\ -2\mathbf{s}_1 & \cdots & -2\mathbf{s}_n \end{bmatrix} \begin{bmatrix} \frac{1}{C_{\mathbf{y}_{11}}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{C_{\mathbf{y}_{nn}}} \end{bmatrix} \begin{bmatrix} 1 & -2\mathbf{s}_1^T \\ \vdots & \vdots \\ 1 & -2\mathbf{s}_n^T \end{bmatrix} \tag{3.46}$$

and after multiplying the three matrices we get

$$\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{A} = \begin{bmatrix} \sum_{i=1}^n \frac{1}{C_{\mathbf{y}_{ii}}} & -2\sum_{i=1}^n \frac{\mathbf{s}_i^T}{C_{\mathbf{y}_{ii}}} \\ -2\sum_{i=1}^n \frac{\mathbf{s}_i}{C_{\mathbf{y}_{ii}}} & 4\sum_{i=1}^n \frac{\mathbf{s}_i\mathbf{s}_i^T}{C_{\mathbf{y}_{ii}}} \end{bmatrix}. \tag{3.47}$$

Now we will calculate the same inverse matrix but only for one agent of the network. So we have

$$\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1}\mathbf{A}_i = \begin{bmatrix} 1 \\ -2\mathbf{s}_i \end{bmatrix} C_{\mathbf{y}_{ii}}^{-1} \begin{bmatrix} 1 & -2\mathbf{s}_i \end{bmatrix} \tag{3.48}$$

and after multiplying the vectors we get

$$\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1}\mathbf{A}_i = \begin{bmatrix} \frac{1}{C_{\mathbf{y}_{ii}}} & -2\frac{\mathbf{s}_i^T}{C_{\mathbf{y}_{ii}}} \\ -2\frac{\mathbf{s}_i}{C_{\mathbf{y}_{ii}}} & 4\frac{\mathbf{s}_i\mathbf{s}_i^T}{C_{\mathbf{y}_{ii}}} \end{bmatrix}. \tag{3.49}$$

So now we know how each agent contributes to the covariance matrix of the estimator. We can do the same for the estimator $\hat{\mathbf{x}}$ since we only have to prove that $\mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{y}$ can be decomposed the same way as presented above. Using $\mathbf{y}$ defined in (3.5) we have

$$\begin{gathered} \mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{y} = \begin{bmatrix} 1 & \cdots & 1 \\ -2\mathbf{s}_1(t) & \cdots & -2\mathbf{s}_n(t) \end{bmatrix} \begin{bmatrix} \frac{1}{C_{\mathbf{y}_{11}}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{C_{\mathbf{y}_{nn}}} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \Leftrightarrow \\ \mathbf{A}^T\mathbf{C}_{\mathbf{y}}^{-1}\mathbf{y} = \begin{bmatrix} \sum_{i=1}^n \frac{y_i}{C_{\mathbf{y}_{ii}}} \\ -2\sum_{i=1}^n \frac{\mathbf{s}_i y_i}{C_{\mathbf{y}_{ii}}} \end{bmatrix}. \end{gathered} \tag{3.50}$$

Computing the contribution of one agent to that same quantity gives

$$\begin{gathered} \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} y_i = \begin{bmatrix} 1 \\ -2\mathbf{s}_i \end{bmatrix} C_{\mathbf{y}_{ii}}^{-1} y_i \Leftrightarrow \\ \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} y_i = \begin{bmatrix} \frac{y_i}{C_{\mathbf{y}_{ii}}} \\ -2\frac{\mathbf{s}_i y_i}{C_{\mathbf{y}_{ii}}} \end{bmatrix}. \end{gathered} \tag{3.51}$$

So here comes a key fact that will motivate our distributed scheme: we conclude that the MVU linear estimator and the covariance matrix can both be computed using a sum of the contribution from every agent as

$$\hat{\mathbf{x}} = \left(\sum_{i=1}^n \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i \right)^{-1} \sum_{i=1}^n \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} y_i \tag{3.52}$$

$$\mathbf{C}_{\hat{\mathbf{x}}} = \left(\sum_{i=1}^{n} \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i \right)^{-1}, \tag{3.53}$$

where each agent only has to provide its own line of the matrix **A**, its own element of the matrix $\mathbf{C_y}$ and its own range measurement from **y**.

Now, as in [28] we will define the matrix $\boldsymbol{\mathcal{P}}_i(\tau)$ and the vector $\mathbf{z}_i(\tau)$ which are both computed by agent i at consensus iteration τ and are used to compute the estimator and the covariance matrix for that agent. We have

$$\hat{\mathbf{x}}_i(\tau) = \boldsymbol{\mathcal{P}}_i^{-1}(\tau)\mathbf{z}_i(\tau) \tag{3.54}$$

$$\mathbf{C}_{\hat{\mathbf{x}}_i(\tau)} = \boldsymbol{\mathcal{P}}_i^{-1}(\tau), \tag{3.55}$$

where we define

$$\boldsymbol{\mathcal{P}}_i(\tau+1) = \frac{\tau}{\tau+1} \sum_{j \in \mathcal{N}_i} \mathcal{W}_{ij} \boldsymbol{\mathcal{P}}_j(\tau) + \frac{1}{\tau+1} \sum_{j \in \mathcal{N}_i} \mathbf{A}_j^T C_{\mathbf{y}_{jj}}^{-1} \mathbf{A}_j \tag{3.56}$$

$$\mathbf{z}_i(\tau+1) = \frac{\tau}{\tau+1} \sum_{j \in \mathcal{N}_i} \mathcal{W}_{ij} \mathbf{z}_j(\tau) + \frac{1}{\tau+1} \sum_{j \in \mathcal{N}_i} \mathbf{A}_j^T C_{\mathbf{y}_{jj}}^{-1} y_j(\tau+1), \tag{3.57}$$

where $\boldsymbol{\mathcal{P}}$ is a $(d+1) \times (d+1)$ matrix and **z** is a $(d+1) \times 1$ vector, $\mathcal{N}_i$ represents the set of neighbors of agent i and i itself and where $\boldsymbol{\mathcal{W}}$ is the so called weight matrix. Each agent has only its own line of the matrix, i.e., agent i only knows $\boldsymbol{\mathcal{W}}_i$, and this line contains the weights given by the said agent to the agents with which it communicates, based on the communications graph $G(t)$, and to itself. The entries of the line that each agent has, have to add up to 1. Thus, W is a stochastic matrix. Normally the weights given by one agent to all the other communicating agents and to itself are equal, and so, the agent only has to know how many agents are communicating with it and assign the inverse of that number to each of the weights. In the case where, for example, one of the agents is unreliable it is possible to rebalance the weights according to the information provided by each neighbor.

In equation (3.57) it is possible to see that **y** changes at every iteration in the consensus phase. This happens because at every iteration every agent collects new measurements that are introduced in the consensus to reduce the variance of the measurements and improve estimation. To introduce new data in the consensus process is called consensus + innovations. A key novelty of our iterations is not only to add innovations for the $\mathbf{z}_i$ from measurements collected in i, but also from neighboring nodes. Similarly, the second term of the $\boldsymbol{\mathcal{P}}_i$ recursion is not the traditional consensus nor the traditional consensus + innovations. This, as we will see, will give our method a very clear advantage in mean error of the distributed quantities, for finite time.

By inspecting equations (3.54) and (3.55) one may mistakenly think that the second sum in equations (3.56) and (3.57) is enough to compute $\boldsymbol{\mathcal{P}}_i(\tau+1)$ and $\mathbf{z}_i(\tau+1)$. This is true if and only if the graph $G(t)$ is fully connected, meaning that every agent of the network communicates with every other, and so the second sum has information from all agents, needed to compute a good approximation for the

estimator and its covariance matix. When $G(t)$ is not fully connected each agent has to find a way to get the needed information from the agents with which it does not communicate. This propagation of information via consensus happens in the first sum since both $\boldsymbol{\mathcal{P}}_j(\tau+1)$ and $\mathbf{z}_j(\tau+1)$ have information from agents that do not communicate with agent i.

The main difference between our method and the one presented in [28] lies in the introduction of the information of the neighbors in the second sum of equations (3.56) and (3.57). The two fractions before the sums in those equations choose which sum gives the largest contribution to the computation of $\boldsymbol{\mathcal{P}}_i(\tau+1)$ and $\mathbf{z}_i(\tau+1)$. In the initial iterations the second sum has more relevance in order to initialize both quantities using the information that an agent can gather from its neighbors and from itself. When $\tau \gg 0$ the first sum becomes dominant since both quantities already have the information from the agent and its neighbors and we want to diffuse that information to the rest of the network to reach a generalized estimator and covariance matrix.

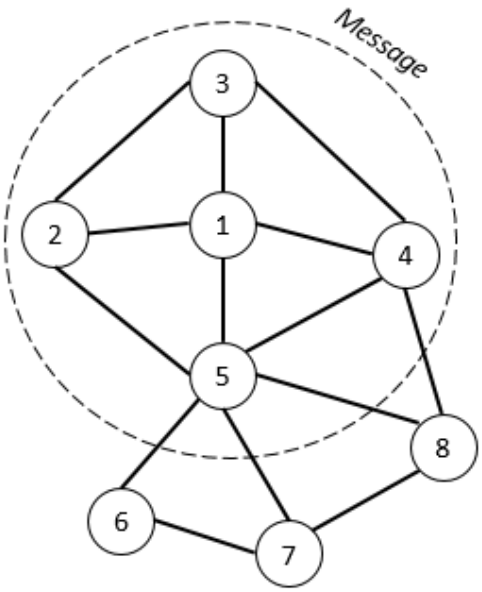


Figure 3.1: Message broadcast from agent 1 on a non fully connected network.

In Figure 3.1 it is possible to observe an example of how the information is spread through agents in a non fully connected graph. The agents communicate through broadcast. This means that an agent, for example 1, broadcasts its message and its neighbors, agents that are in the range of 1, receive it and use it to do their computations. The information broadcast at $\tau = 0$ is $\mathbf{A}_1(1)$, $C_{\mathbf{y}_{11}}(1)$ and $y_1(1)$ since the quantities $\boldsymbol{\mathcal{P}}_1$ and $\mathbf{z}_1$ are not yet computed. Since $\mathbf{A}_1(1) = [1 - 2\mathbf{s}_1^T]$ it only has to broadcast its position $\mathbf{s}_1$. In the remaining iterations, since the configuration of the network and the position of the target does not change, $\mathbf{A}_1(\tau+1)$ and $C_{\mathbf{y}_{11}}(\tau+1)$ do not change, the agent only has to broadcast $y_1(\tau+1)$, $\boldsymbol{\mathcal{P}}_1(\tau)$ and $\mathbf{z}_1(\tau)$.

So the communication process has two stages. The first stage at $\tau = 0$ is seen as an initialization stage, where the number of communications made is given by

$$\#\text{communications}(\tau = 0) = n(d+2). \tag{3.58}$$

The second stage is where the consensus occurs and the number of communications between agents is given by

$$\#\text{communications}(\tau > 0) = nT\Big(\frac{(2+d)(d+1)}{2} + (d+1) + 1\Big), \tag{3.59}$$

where T is the total number of consensus iterations.

Theorem 1. *For each node i the estimator $\hat{\boldsymbol{x}}_i(\tau)$ is centered if*

$$E_{\boldsymbol{x}}[\hat{\boldsymbol{x}}_i(\tau)] = \boldsymbol{x} \ \forall_{\tau,\boldsymbol{x}}, \tag{3.60}$$

where $E_{\boldsymbol{x}}[X]$ is the expected value of a random variable X with respect to the distribution of $\boldsymbol{x}$.

Proof. We start by computing the expected value of $\mathbf{z}_i(\tau+1)$, given by equation (3.57), resulting in

$$E_{\mathbf{x}}[\mathbf{z}_i(\tau+1)] = \frac{\tau}{\tau+1}\sum_{j\in\mathcal{N}_i}\mathcal{W}_{ij}E_{\mathbf{x}}[\mathbf{z}_j(\tau)] + \frac{1}{\tau+1}\sum_{j\in\mathcal{N}_i}\mathbf{A}_j^T C_{\mathbf{y}_{jj}}^{-1}E_{\mathbf{x}}[y_j(\tau+1)], \tag{3.61}$$

since $E_{\mathbf{x}}[\mathbf{y}(\tau+1)] = \mathbf{Ax}$ and considering $\boldsymbol{\mathcal{P}}_i(\tau+1)$, given by equation (3.56), we get

$$E_{\mathbf{x}}[\mathbf{z}_i(\tau+1)] = \boldsymbol{\mathcal{P}}_i(\tau+1)\mathbf{x} \tag{3.62}$$

and considering equation (3.54)

$$\begin{aligned} E_{\mathbf{x}}[\hat{\mathbf{x}}_i(\tau)] &= \boldsymbol{\mathcal{P}}_i^{-1}(\tau)E_{\mathbf{x}}[\mathbf{z}_i(\tau)] \Leftrightarrow \\ E_{\mathbf{x}}[\hat{\mathbf{x}}_i(\tau)] &= \boldsymbol{\mathcal{P}}_i^{-1}(\tau)\boldsymbol{\mathcal{P}}_i(\tau)\mathbf{x} \Leftrightarrow \\ E_{\mathbf{x}}[\hat{\mathbf{x}}_i(\tau)] &= \mathbf{x} \ \forall_{\tau,\mathbf{x}}. \end{aligned} \tag{3.63}$$

This proves that the distributed estimator is centered. □

Despite the estimator not being efficient, preliminary simulations show, for finite consensus rounds, that it is faster than other efficient estimators, like the consensus + innovations one. In conclusion, this section contains one of the major contributions from this work since we presented a distributed method where, at each iteration, each agent computes an unbiased estimate using ISEE.U. Further analysis is required to rebalance the variance of the estimator and better assess the finite and asymptotic behavior of $\tau\text{var}(\mathbf{x}_i(\tau))$. This is left for future work.

In Algorithm 1 we present a pseudocode for the ISEE.U estimator. The input comprises the matrices and vectors necessary to compute the estimator and its convariance matrix and the position of the target and standard deviation of all agents that are used to compute the innovations as in equation (3.57). It is also necessary to introduce the total number of consensus iterations T and an adjacency matrix $\boldsymbol{\mathcal{A}}$. This matrix describes the edges of the communications graph G since each entry can only be 0 or 1 depending on whether a communication between the two corresponding agents exists or not. This

matrix is symmetric. The output of the algorithm is the estimator $\hat{\mathbf{x}}$ and its covariance matrix $C_{\hat{\mathbf{x}}}$. The algorithm starts by computing $\mathcal{P}_i$ and $\mathbf{z}_i$ for all agents at $\tau = 0$ from the quantities defined as input. In step 4 we transform the adjacency matrix into the weight matrix, giving equal weights to all agents within the same line. In steps 6 and 7 we define the computed quantities as the previous matrix and vector for the first iteration of consensus. Then the consensus iterations start, where $\mathcal{P}_i(\tau + 1)$ and $\mathbf{z}_i(\tau + 1)$ are updated for each agent according to equations (3.56) and (3.57), respectively, until T is reached, new range measurements $\mathbf{y}(\tau + 1)$ are also acquired by each agent as in equation (3.5) that are used in the computations of $\mathbf{z}_i(\tau + 1)$. For each τ, $\mathcal{P}_i(\tau)$ and $\mathbf{z}_i(\tau)$ are also updated to save the value of the current matrix and vector to be used in the next iteration. Lastly, the for loop in step 16 computes the ISEE.U estimator and corresponding covariance matrix for each agent. As the goal of consensus is to reach a value that is agreed upon for the entire network, we can return the value from any agent since they are approximately equal.

3.4 Network Control

Now that we described the localization part of our method we have to detail the other contribution of this thesis for addressing the active target localization problem. Since we are working with a network of mobile agents it is necessary to control the movement of the agents on that network. This can be done in various ways but the topic of this thesis is to move our network in order to improve the estimation accuracy of the target position. So we consider a random deployment of the network's nodes and at each iteration each agent moves to a position where the localization error is minimized. In order to do so we will use the covariance matrix of the estimator for each agent $\mathbf{C}_{\hat{\mathbf{x}}_i}$ defined in (3.55).

As explained in Section 3.2 the covariance can be used as a measure of how dispersed the data is along specific directions. In the case of the ISEE.U estimator, the covariance matrix indicates how scattered the estimations $\hat{\mathbf{x}}$ will be along the multiple directions, giving us a measure of dispersion for the estimates. Since $\hat{\mathbf{x}}$ follows a normal distribution, as seen in (3.30), it is possible to see that the estimates become more scattered when they are further away from the real value of $\mathbf{x}$, and thus that the estimation accuracy is not the best. In the literature, many authors choose to use this same approach but using the FIM. The FIM [23] is a tool to measure the amount of information that a random variable has about an unknown parameter of a distribution that models it and it is defined as $\mathbf{I} = \mathbf{C}_{\hat{\mathbf{x}}}^{-1}$. So using the FIM our goal is to maximize the information that we have about the estimator in order to produce more accurate estimates, which is the inverse process in comparison with minimizing the error. We decided to use the covariance matrix, also called precision matrix, because its effects on the estimation process are easier to understand. Then, the goal of ISEE.Uctl is to minimize the estimation error from the covariance matrix of the estimator.

According to [29] there are three traditional criteria used to solve these types of minimization problems. These are the *A-optimality* criterion, *D-optimality* criterion and the *E-optimality* criterion. The *A-optimality* criterion consists on minimizing the trace of the covariance matrix, the *D-optimality* criterion seeks to minimize the determinant of that matrix, which is the same as minimizing the volume of

the error ellipsoid, and finally the *E-optimality* criterion tries to minimize the maximum eigenvalue of the covariance.

The *D-optimality* criterion was the chosen one since it considers an error ellipsoid, whose axes are obtained from the eigenvalues of the covariance matrix, and tries to minimize its volume, which entails the minimization of the length of all axes which, in turn, is the same as minimizing all the eigenvalues of the covariance matrix. This criterion has only one problem: if one of the eigenvalues is zero, the axis of the ellipsoid associated with this eigenvalue has zero length and its volume becomes zero, even though we can have large errors in the directions associated with the remaining eigenvalues. One solution is to implement this criterion with the *E-optimality* criterion at the same time. The *D-optimality* criterion is now used in the majority of the situations but if one or more of its eigenvalues become zero we use the *E-optimality* criterion. However, if all eigenvalues are zero then the volume of the error ellipsoid is zero and we have perfect accuracy of estimation. In this situation, the *D-optimality* criterion is still used. We should stress, however, that a zero eigenvalue has never appeared in our extensive simulations. Actually, we believe the occurence of such an event is highly unprobable, so the *E-optimality* criterion is only implemented as a security method.

In order to make the *D-optimality* criterion work we have to know how to convert the value of the eigenvalues from the covariance matrix into the length of the axis of the ellipsoid. This is done by using

$$l_j = \sqrt{\chi^2_{(d+1),\theta}\lambda_j} \tag{3.64}$$

where l_j is half of the length of the axis associated with the eigenvalue λ_j. The $\chi^2_{(d+1),\theta}$ is the value for the chi-squared distribution with $d+1$ degrees of freedom and a certain confidence level.

The chi-squared distribution is the distribution of the sum of squares of independent random variables, where the number of these random variables is equal to the degrees of freedom. This distribution is very useful to build confidence intervals [22]. The use of this distribution in the computation of the axes lengths is appropriate because for a certain confidence interval we know how many estimates will be inside our error ellipsoid. For example, using a confidence interval of 95% means that at least 95% of the estimates for a given position of the network will be inside our error ellipsoid. The confidence interval is implicit in the θ factor and for a 95% confidence interval we have $\theta = 0.05$.

Lastly, the volume of the ellipsoid is given by

$$\begin{aligned} V_{2\epsilon} &= \frac{\pi^\epsilon}{\epsilon!}\prod_{j=1}^{2\epsilon} l_j \\ V_{2\epsilon+1} &= \frac{2(\epsilon!)(4\pi)^\epsilon}{(2\epsilon+1)!}\prod_{j=1}^{2\epsilon+1} l_j \end{aligned} \tag{3.65}$$

where 2ϵ or $2\epsilon+1$ is the dimension of the ellipsoid. This dimension should be equal to the dimension of the covariance matrix of the estimator since the number of eigenvalues of this matrix is equal to its dimension.

Considering that the vector of solutions **x** has always one more dimension than the space we are working in, the ellipsoid has that one extra dimension too. So, if our deployment area is in $\mathbb{R}^2$, the

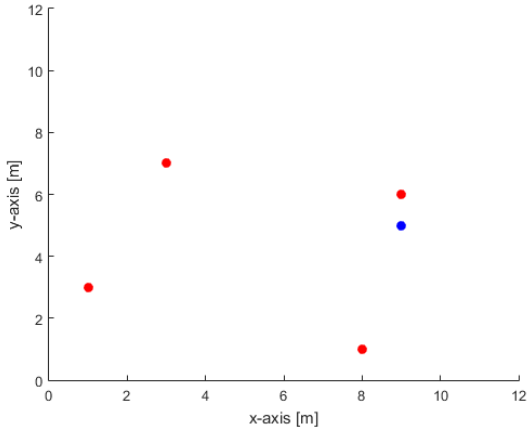
ellipsoid will be in $\mathbb{R}^3$. For this reason it is difficult to understand a graphic representation of the network and the ellipsoid together since they have different dimensions.

To address this problem, and only to give an example of a graphical representation of the ellipsoid and the estimated points, we will use a simpler linear model where the dimension of **x** is the same of the deployment area. So consider a setup of a network with 4 agents and 1 target and a problem formulated by the linear model given by (3.11) but where

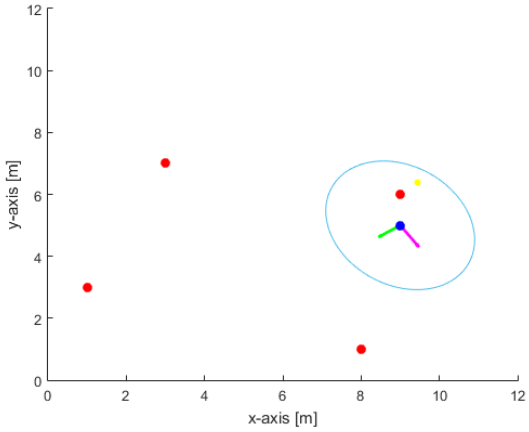
$$\mathbf{y} = \begin{bmatrix} \frac{1}{2}(r_2^2 - r_1^2 - ||\mathbf{s}_1 - \mathbf{s}_2||^2) + (\mathbf{s}_1 - \mathbf{s}_2)^T\mathbf{s}_1 \\ \vdots \\ \frac{1}{2}(r_{i+1}^2 - r_1^2 - ||\mathbf{s}_1 - \mathbf{s}_{i+1}||^2) + (\mathbf{s}_1 - \mathbf{s}_{i+1})^T\mathbf{s}_1 \\ \vdots \\ \frac{1}{2}(r_n^2 - r_1^2 - ||\mathbf{s}_1 - \mathbf{s}_n||^2) + (\mathbf{s}_1 - \mathbf{s}_n)^T\mathbf{s}_1 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} (\mathbf{s}_1 - \mathbf{s}_2)^T \\ \vdots \\ (\mathbf{s}_1 - \mathbf{s}_{i+1})^T \\ \vdots \\ (\mathbf{s}_1 - \mathbf{s}_n)^T \end{bmatrix} \tag{3.66}$$

$$\mathbf{x} = \mathbf{p}.$$

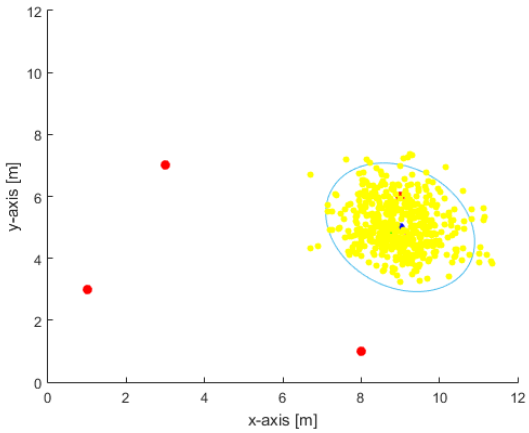
In Figure 3.2b it is possible to observe an error ellipsoid in 2D and the direction vectors of the major and minor axes, which correspond to the largest and smallest eigenvalues of the estimator's covariance matrix, respectively. In Figure 3.2c we plotted 500 target position estimates, which allow to draw some conclusions about the relation between those estimates and the error ellipsoid. As you can see, many estimates fall inside the error ellipsoid, which was expected since a 95% confidence interval was used, so in order to approach those estimates and the real target position we have to make the ellipsoid smaller. This results in more accurate estimates.



(a) Initial setup of the network.



(b) Error ellipsoid.



(c) Target position estimates.

Figure 3.2: Relation between the error ellipsoid and the target position estimates. The red dots represent the agents, the blue dot is the real target position, the yellow dots are the target position estimates, the error ellipsoid is presented in light blue and the purple and green arrows represent the major and minor axes' direction vectors, respectively. Most of the target position estimates fall inside the error ellipsoid.

Finally, the procedure used by agent i to compute the motion control is the following:

1. Compute the covariance matrix $\mathbf{C}_{\tilde{\mathbf{x}}}$ through consensus;
2. Compute the volume for the current position of the network;
3. Subtract the agent contribution $(\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i)^{-1}$ from the covariance matrix and save the resulting matrix;
4. Choose the new position;
5. Compute $(\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i)^{-1}$ for the new position;
6. Add its contribution to the resulting matrix from step 3;
7. Compute the new volume;

8. Compare the two volumes, if the new volume is smaller then move the agent to the new position, if it is larger then keep the agent in its old position.

To make this solution realistic we discretize the deployment area. When an agent wakes up and wants to move, it follows the presented procedure from step 4 for all the eight possible positions around its current one. This is a greedy algorithm since each agent makes the locally optimal choice every time it wants to move, hoping that in the end the network can reach a global minimum.

Algorithm 2 is the conversion of the procedure presented for the control of one agent into pseudocode. Firstly we defined as input $\mathbf{A}_i$, $C_{\mathbf{y}_{ii}}$, $\mathbf{C}_{\hat{\mathbf{x}}}$ previously obtained by consensus, the dimension of the space which can also be obtained from $\mathbf{A}$, and the chi-squared function value for the desired confidence interval. The output of the algorithm will be the next position of the agent. The procedure previously presented can explain the content of the pseudocode since they follow the same order of events. However, some little steps in the code may not be very explicit. In step 6 matrix $\mathbf{C}_{RS}$ is the covariance matrix for the remaining agents which does not contain the contribution of the agent which is moving. The test presented in step 8 is necessary because two agents or an agent and a target can not occupy the same position at the same time. So it is necessary to check if that position is free before considering the move to that position. In steps 10 to 12 the quantities that are provided as input are again computed for the new position, creating temporary variables. If the new volume is smaller than the previous one the values from these temporary variables will replace the values for our input variables, which happens in steps 16 and 17. Since the loop only ends when all the 8 positions are tested (if they are available), every time the volume is minimized we have to save that position as the best one and compare that position with the remaining ones. This is why we save the new volume and new matrix $\mathbf{C}_{RS}$ in steps 18 and 19 respectively.

If one wants to implement the *E-optimality* criterion as a security method in the algorithm, another variable has to be created that always stores the maximum eigenvalue of the matrix each time the volume is computed. Then before step 2 a test is performed where if there are no eigenvalues equal to zero or if all the eigenvalues are equal to zero it uses the volumes as the criterion. Else, if we have eigenvalues equal to zero and others that are not zero we have to use the minimization of the maximum eigenvalue as the criterion.

In conclusion, we now have presented a way for the agents to move and reconfigure the network, where the control for the movement is computed considering the maximization of the accuracy of estimation of the target position.

Algorithm 1 ISEE.Uest: Distributed estimation

Input: $\mathbf{A}$;
$\mathbf{y}$;
$\mathbf{C}_{\mathbf{y}}$;
$\boldsymbol{\mathcal{A}}$; // Adjacency matrix
T; // Number of consensus iterations
$\mathbf{p}$; // Target position
$\boldsymbol{\sigma}$; // Vector containing the standard deviation of all agents

Output: $\hat{\mathbf{x}}$;
$\mathbf{C}_{\hat{\mathbf{x}}}$;

for each agent involved i **do**
  $\boldsymbol{\mathcal{P}}_i^{(1)} = \sum_{j \in \mathcal{N}_i} \mathcal{A}_{ij} \mathbf{A}_j^T C_{\mathbf{y}_{jj}}^{-1} \mathbf{A}_j$;
  $\mathbf{z}_i^{(1)} = \sum_{j \in \mathcal{N}_i} \mathcal{A}_{ij} \mathbf{A}_j^T C_{\mathbf{y}_{jj}}^{-1} y_j$;
  each agent assigns the weights from $\boldsymbol{\mathcal{A}}$
  $\boldsymbol{\mathcal{W}}_i = \boldsymbol{\mathcal{A}}_i \left(\boldsymbol{\mathcal{A}}_i \mathbf{1} \right)^{-1}$; // $\mathbf{1}$ is a vector of 1s with size $\mathrm{size}(\boldsymbol{\mathcal{A}}_i) \times 1$.
end for
$\boldsymbol{\mathcal{P}}^{(\tau)} = \boldsymbol{\mathcal{P}}^{(0)}$;
$\mathbf{z}^{(\tau)} = \mathbf{z}^{(0)}$;
while the number of consensus iterations T is not met, each agent i **do**
  update $\boldsymbol{\mathcal{P}}_i^{(\tau+1)}$ via (3.56) using $\boldsymbol{\mathcal{P}}_i^{(\tau)}$;
  compute new range measurements $y_i^{(\tau+1)}$ via (3.5); // Innovations.
  update $\mathbf{z}_i^{(\tau+1)}$ via (3.57) using $\mathbf{z}_i^{(\tau)}$;
  $\boldsymbol{\mathcal{P}}^{(\tau)} = \boldsymbol{\mathcal{P}}^{(\tau+1)}$;
  $\mathbf{z}^{(\tau)} = \mathbf{z}^{(\tau+1)}$;
  $\tau = \tau + 1$;
end while
for each agent involved i **do**
  $\hat{\mathbf{x}}_i = \left(\boldsymbol{\mathcal{P}}^{(\tau)} \right)^{-1} \mathbf{z}_i^{(\tau)}$;
  $\mathbf{C}_{\hat{\mathbf{x}}_i} = \left(\boldsymbol{\mathcal{P}}_i^{(\tau)} \right)^{-1}$;
end for
return $\hat{\mathbf{x}} = \hat{\mathbf{x}}_i$;
$\mathbf{C}_{\hat{\mathbf{x}}} = \mathbf{C}_{\hat{\mathbf{x}}_i}$;

Algorithm 2 ISEE.Uctl: Distributed agent control

Input: $\mathbf{C}_{\hat{\mathbf{x}}}$;
$\mathbf{A}_i$;
$C_{\mathbf{y}_{ii}}$;
d; // Dimension of the space
$\chi^2_{(d+1),\theta}$; // Chosen chi-squared value

Output: $\mathbf{s}_i(t+1)$;

1: calculate the eigenvalues of $\mathbf{C}_{\hat{\mathbf{x}}}$ using
$det(\mathbf{C}_{\hat{\mathbf{x}}} - \lambda \mathbf{I}) = 0$;
2: **for** all the eigenvalues of $C_{\hat{\mathbf{x}}}$ **do**
3: compute the lengths of the axis of the ellipsoid
$l_j = \sqrt{\chi^2_{(d+1),\theta}\lambda_j}$
4: **end for**
5: compute the volume for the current position $V(t)$ via (3.65) considering $d+1$;
6: $\mathbf{C}_{RS} = \mathbf{C}_{\hat{\mathbf{x}}} - (\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i)^{-1}$;
7: **for** all 8 discrete positions around the current one **do**
8: check if the new position is occupied by other agent or target;
9: **if** the position is free **then**
10: $\mathbf{s}_{test} = new\_position$;
11: compute $\mathbf{A}_{test}$ and $\mathbf{C}_{\mathbf{y}_{test}}$ for the new position;
12: $\mathbf{C}_{\hat{\mathbf{x}}_{test}} = \mathbf{C}_{RS} + (\mathbf{A}_{test}^T C_{\mathbf{y}_{test}}^{-1} \mathbf{A}_{test})^{-1}$;
13: repeat the for loop in 2 using $\mathbf{C}_{\hat{\mathbf{x}}_{test}}$;
14: repeat step 5 which gives the volume for the new position V_{test};
15: **if** $V_{test} < V(t)$ **then**
16: $\mathbf{s}_i = \mathbf{s}_{test}$, $\mathbf{A}_i = \mathbf{A}_{test}$, $C_{\mathbf{y}_{ii}} = C_{\mathbf{y}_{test}}$;
17: $\mathbf{C}_{\hat{\mathbf{x}}} = \mathbf{C}_{\hat{\mathbf{x}}_{test}}$;
18: $V(t+1) = V_{test}$;
19: $\mathbf{C}_{RS} = \mathbf{C}_{\hat{\mathbf{x}}} - (\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} \mathbf{A}_i)^{-1}$;
20: **end if**
21: **end if**
22: **end for**
23: **return** $\mathbf{s}_i(t+1) = \mathbf{s}_i$;

Chapter 4

Extensions

In this chapter we will present some solutions to problems that can arise from the main localization problem considered in Chapter 3.

These extensions are needed to scale the main problem and solve a greater number of problems that come from the generic one without having to redo all the mathematics involved.

4.1 Multiple Targets

The first problem we will address in this chapter is the simultaneous localization of multiple targets. As mentioned in Chapter 1, our method can be used to locate multiple targets, but in Chapter 3 we only considered one target in order to simplify the computations involved in the process. Now we will show how the method ISEE.U from Chapter 3 can be generalized.

Starting by the linear model presented in (3.6), adding more targets does not change the model itself, we only need to change the definitions of the matrices in it. Considering a network of n agents and K targets our solution vector now has the positions of all the targets that are part of the network and our variable **x** is now

$$\mathbf{x} = \begin{bmatrix} ||\mathbf{p}_1||^2 \\ \mathbf{p}_1 \\ \vdots \\ ||\mathbf{p}_k||^2 \\ \mathbf{p}_K \end{bmatrix}, \tag{4.1}$$

a $K(d+1) \times 1$ vector.

The measurements vector **y** will now have to contain the measurements from all agents to all targets. The vector structure remains the same, we only need to take all the $\mathbf{y}_k$ vectors for all K targets and stack them as

$$\mathbf{y} = \begin{bmatrix} r_{11}^2 - ||\mathbf{s}_1||^2 \\ \vdots \\ r_{n1}^2 - ||\mathbf{s}_n||^2 \\ r_{12}^2 - ||\mathbf{s}_1||^2 \\ \vdots \\ r_{nK}^2 - ||\mathbf{s}_n||^2 \end{bmatrix}, \tag{4.2}$$

and since the noise is related to the measurements, both vectors should have the same structure , namely,

$$\mathbf{w} = \begin{bmatrix} w_{11} \\ \vdots \\ w_{n1} \\ w_{12} \\ \vdots \\ w_{nK} \end{bmatrix}, \tag{4.3}$$

where both vectors are $nK \times 1$.

The **A** matrix definition depends on how the solution vector is arranged. The matrix lines have the same information but we need to add blocks of zeros to the matrix in order to adjust its size to the size of the new problem,

$$\mathbf{A} = \begin{bmatrix} 1 & -2\mathbf{s}_1^T & & & \\ 1 & -2\mathbf{s}_n^T & & 0 & \\ & & \vdots & & \\ & 0 & & 1 & -2\mathbf{s}_1^T \\ & & & 1 & -2\mathbf{s}_n^T \end{bmatrix}, \tag{4.4}$$

and **A** is now a $nK \times K(d+1)$ block-diagonal matrix.

Lastly, for the Ξ matrix, defined in (3.8), it is only necessary to repeat the diagonal of the matrix as many times as the number of targets,

$$\Xi = \begin{bmatrix} 2||\mathbf{s}_1 - \mathbf{p}_1|| & & & & \\ & \ddots & & 0 & \\ & & 2||\mathbf{s}_n - \mathbf{p}_1|| & & \\ & 0 & & \ddots & \\ & & & & 2||\mathbf{s}_n - \mathbf{p}_K|| \end{bmatrix}, \tag{4.5}$$

which is now a $nK \times nK$ matrix

As an example, if we are working in $\mathbb{R}^2$ with a network comprising 4 agents and 3 targets, **A** will be a 12×9 matrix, Ξ will be a 12×12 matrix, **y** and **w** will both be 12×1 vectors and **x** will be a 9×1 vector.

The last input that is not yet defined in this section is the inverse covariance matrix of the measurement data $\mathbf{C}_{\mathbf{y}}^{-1}$ considering the method in Section 3.2.1. This matrix is still diagonal and the only transformation needed is to add to its diagonal the covariance relative to all the targets, ending up with

$$\mathbf{C}_{\mathbf{y}}^{-1} = \begin{bmatrix} \frac{1}{4||\mathbf{s}_1-\mathbf{p}_1||^2\sigma_{11}^2} & & & & \\ & \ddots & & 0 & \\ & & \frac{1}{4||\mathbf{s}_n-\mathbf{p}_1||^2\sigma_{n1}^2} & & \\ & 0 & & \ddots & \\ & & & & \frac{1}{4||\mathbf{s}_n-\mathbf{p}_K||^2\sigma_{nK}^2} \end{bmatrix}, \tag{4.6}$$

which is a $nK \times nK$ matrix. For the other method presented in Section 3.2.1 the extension of the matrix is analogous.

Now that the input data is defined it is necessary to explain how the consensus is performed in order to compute the ISEE.U estimator and to move the network. This process is based on the same steps as presented in Section 3.3, but more information should be processed and broadcast by the agents.

Recalling the $\mathcal{P}$ matrix and **z** vector from (3.56) and (3.57), respectively, each agent can now compute these quantities independently for each target. This means that agent i should compute K matrices $\mathcal{P}$ and vectors **z** and compute the ISEE.U estimator $\hat{\mathbf{x}}_{ik}$ and covariance matrix of the estimator $\mathbf{C}_{\hat{\mathbf{x}}_{ik}}$ at consensus iteration τ as

$$\hat{\mathbf{x}}_{ik}(\tau) = \mathcal{P}_{ik}^{-1}(\tau)\mathbf{z}_{ik}(\tau), \tag{4.7}$$

$$\mathbf{C}_{\hat{\mathbf{x}}_{ik}}(\tau) = \mathcal{P}_{ik}^{-1}(\tau). \tag{4.8}$$

The message broadcast by each agent also changes slightly. In this new configuration agent i broadcasts at $\tau = 0$ its own position $\mathbf{s}_i$ and a matrix containing the range measurements and the respective covariance matrix entry given by

$$\begin{bmatrix} y_{i1} & C_{\mathbf{y}_{ii}} & 1 \\ & \vdots & \\ y_{iK} & C_{\mathbf{y}_{(iK)(iK)}} & K \end{bmatrix}. \tag{4.9}$$

The data contained in the third column of this matrix works as a label for other agents to know to which estimator the data should be assigned. The number of communications during the initialization stage is now

$$\#\text{communications}(\tau = 0) = n(d + 3K). \tag{4.10}$$

In the second stage, where consensus occurs, each agent only has to broadcast one matrix $\boldsymbol{\mathcal{P}}(\tau)$, one vector $\mathbf{z}(\tau)$, and the new range measurement $\mathbf{y}_{ik}(\tau+1)$ for each target with the respective label. The number of communications in this stage is given by

$$\#\text{communications}(\tau > 0) = nKT\Big(\frac{(2+d)(d+1)}{2} + (d+1) + 2\Big), \tag{4.11}$$

where T is again the total number of consensus iterations.

The only thing left is to generalize the agent movement procedure for multiple targets. Because each target can compute the covariance matrix of the estimator for each target independently, it can also compute the volumes of the K error ellipsoids separately, since we will use the *D-optimality* criterion as in Section 3.4. The procedure presented in that section should be done for each target separately and only the step where the volumes are compared changes to include all the volumes for all the targets. To compare the volumes, one has several possible approaches. We choose to minimize the combined accuracy of estimation of the K target positions. In this case, the movement of the network may increase the size of one ellipsoid if it improves the general accuracy. This approach is not always used because one may want to improve the accuracy of estimation for only a certain target position, or assign certain agents to a certain target or group of targets.

For the chosen approach we have to find a way to minimize all the volumes at the same time. To do so we choose to minimize the cost function defined as the arithmetic mean of all volumes considered, given by

$$J(V) = \frac{1}{K}\sum_{k=1}^{K} V_k. \tag{4.12}$$

Since the volume is basically the multiplication of the axes' lengths and this is computed from the eigenvalues of the covariance matrix, minimizing the average between all volumes is the same as minimizing all the eigenvalues from the K covariance matrices at the same time. This method also guarantees that volumes equal to zero do not make it invalid. As an example, imagine that we are considering 4 targets and changing the position of the agent results in the volume of the error ellipsoid for one target being zero. If we consider the geometric mean as the cost function, when the agent changes position the cost function will be zero and the agent will consider this position as the best one, which is a wrong decision. With the average mean, the cost function would still have a positive value and the position where the volume is zero would not necessarily beat the previous one. This method also guarantees that if an agent looses a target from its range due to the change of position, the new cost function can still be comparable to the previous one since the average makes possible to compare cost functions computed with different volumes.

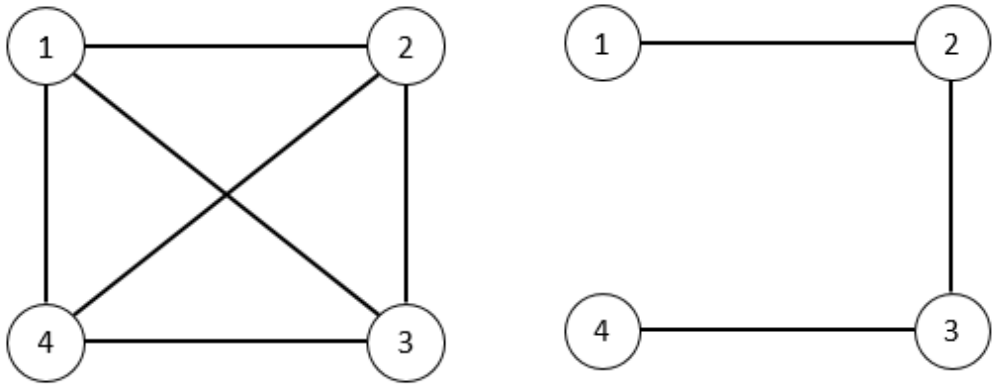
As in Section 3.4 we still have the problem where we can have eigenvalues equal to zero. In those cases we can again use the *E-optimality* criterion. This is not the best criterion because it only minimizes the largest eigenvalue. Since we are trying to improve the general accuracy of estimation, we have to consider all the eigenvalues from all the K covariance matrices and only minimizing the largest one brings more sources of possible minimization errors.

4.2 Incomplete Graphs

In this section we will explain how our ISEE.U method solves problems where the communications graph $G(t)$ or measurements graph $\mathcal{G}(t)$ are incomplete.

4.2.1 Incomplete Communications Graph

The communications graph becomes incomplete (or non fully connected) when at least one of the agents of the network moves out of the broadcast range from the other agents. This subject was already mentioned when explaining the ISEE.U estimator method in Section 3.3 but it will be studied in more detail in this section.



(a) Fully connected communications graph. **(b)** Incomplete communications graph - *path*.

Figure 4.1: Examples of a fully connected communications graph and an incomplete one. The circles represent the agents. The example presented for the incomplete graph is a special case called *path*.

As mentioned in section 3.3 the method is robust to incomplete communications graph where all the agents are connected to each other by at least one edge, but the diameter of the network is larger than one. This is what happens in Figure 4.1b. In this case the iterations of matrices $\mathcal{P}_i$ and vectors $\mathbf{z}_i$ guarantee the diffusion of information between all agents. This means that for the graph presented in Figure 4.1b, the information that agent 4 has about the target will be passed by $\mathcal{P}_4$ and $\mathbf{z}_4$ to agent 3 which will pass this information to agent 2 through $\mathcal{P}_3$ and $\mathbf{z}_3$ and so on until it reaches agent 1. When agent 1 produces an estimate for the position of the target it will consider information that comes from agent 4 even though 1 only broadcast to agent 2.

However, one exception arises that the consensus cannot handle. This case is when an agent looses the communication with the rest of the network. Since it cannot receive information from the other agents it can only use its own measurements to compute an estimate of the target position. This degenerate scenario can produce very noisy estimates. There are three approaches that one can follow to solve this problem:

- The agent moves back to its previous position in order to try to get back to the network;

- The agent tries to predict what is the current position of the network since it knows the previous positions of the agents that were broadcast in the last consensus phase;
- The agent stops moving and waits for the network to move to a position near it and only when it is back inside the network it can start moving again;
- The agent does a *Random Walk*, moving in random directions, until it finds the network.

Every approach has its advantages and disadvantages but the one that is generally used in the literature is the *Random Walk*.

Finally, seeing the special case that arises from one agent being disconnected from the network, one may ask what happens when it happens to a group of agents. This situation is a lot easier to handle than the previous one. In the previous case, only one agent is disconnected from the network so it has to rely in its own measures to provide the estimate for the target position. On the other hand if a group of agents is disconnected from the network it can form another network and perform consensus only with the agents of that network.

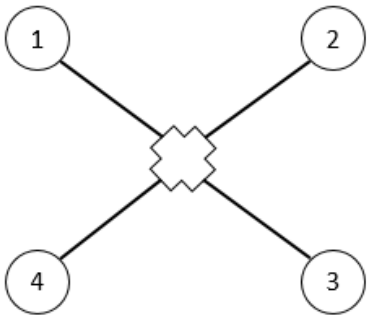
4.2.2 Incomplete Measurements Graph

The measurements graph becomes incomplete when at least one agent does not have range to get range measurements from one of the targets.

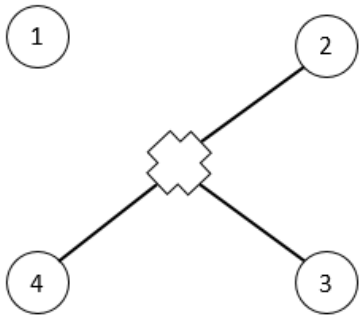
A case where the measurements graph is incomplete can be solved by ignoring the information given by an agent that cannot measure range. The measurements vector entry for agent i that leaves the graph is given by $y_i = r_i^2 - ||\mathbf{s}_i||^2$ and since it cannot measure range $r_i = 0$, $y_i = -||\mathbf{s}(t)_i||^2$. In this case, $y_i \neq 0$ and the information broadcast by this agent will still be incorporated into the consensus phase which may lead to wrong estimates. The approaches found to prevent this issue are:

- When an agent cannot perform range measurements it does not broadcast information so the other agents do not receive it, and so they will perform consensus with only the information from the other agents;
- When an agent cannot perform range measurements it broadcasts the usual information but on the initialization phase it communicates one extra number indicating that its information is corrupted. This way the other agents know what information they should not use and adjust their weight matrix lines to mark the weight relative to the corrupt agent as 0 and the other weights as if the network had less agents.

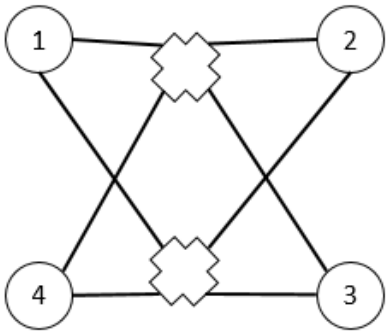
The second approach has advantages when dealing with more than one target because if one or more agents cannot measure the range relative to one target they still should broadcast information since they may contribute for the localization of the targets to which they can measure the range. In this case instead of selecting which information to broadcast it might be preferable to send a tag that can be a natural number along with the corrupt data so that the other agents can deal with the information appropriately.



(a) Fully connected measurements graph.



(b) Incomplete measurements graph.



(c) Fully connected measurements graph with 2 targets.

Figure 4.2: Examples of a fully connected measurements graphs with one and two targets and an incomplete one. The targets are represented by the crosses while the circles represent the agents.

On the other hand, the consensus phase allows an agent that has no measurements relative to a target to accurately estimate its position. As seen for the communications graph, consensus allows the agents to share information and thus obtain measurements from the rest of the network. If an agent i which cannot perform range measurements ignores its own contribution to the matrix $\mathcal{P}_i$ and vector $\mathbf{z}_i$ it can - through the information from connected neighbors - accurately estimate the position of a target that it cannot even measure.

Chapter 5

Numerical Results

In this Chapter our method presented in Chapter 3 will be extensively tested in order to validate its performance experimentally.

Firstly, we determine experimentally the number of consensus needed in order to obtain close approximations to the centralized matrices and experimentally compare distributed estimators as mentioned in Section 3.3, then we will compare the ISEE.U estimator with the biased linear estimator presented in Section 3.2. Then two methods to improve the position estimates will be presented. Finally, simulations of the ISEE.U will be presented in different scenarios and the results will be compared to the ones obtained by the benchmark described in Section 2.2.

5.1 Distributed algorithm

In this section we start by determining experimentally the minimum number of consensus iterations T needed in order to obtain good approximations to the centralized method. We try to estimate the minimum number of iterations because these iterations increase the number of communications between agents, as you can see in Section 3.3, which can be an inconvenient since during the communications the agents are spending battery, which can be limited, and time.

In order to do so, a test was designed which deployed the agents and a static target randomly in a square and then it computed $\mathcal{P}$ and $\mathbf{z}$ for a number of total iterations from 1 to 100. In the case of $\mathcal{P}$, the algorithm computes the volume for each agent and compares it with the volume obtained for the centralized case. Then it saves an average of the difference between volumes for all agents for each number of T. For $\mathbf{z}$ it does a similar procedure but the $\mathbf{z}$ vectors are compared using the *Root Mean Squared Error* (RMSE) given by

$$\mathrm{RMSE}(T) = \sqrt{\frac{\sum_{j=1}^{M} ||\mathbf{z}_{ML} - \mathbf{z}_j(T)||^2}{M}}, \tag{5.1}$$

where $\mathbf{z}_j$ is the vector obtained for each number of total consensus iterations T, M is the number of Monte Carlo trials performed and $\mathbf{z}_{ML}$ is the vector obtained for the centralized case.

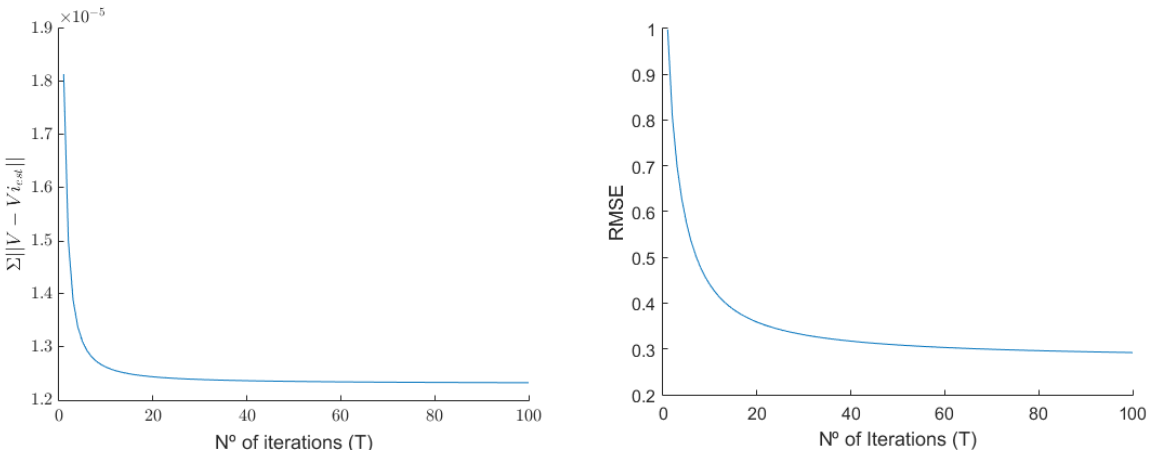
Table 5.1: Minimum number of consensus iterations for each network.

Number of agents	Number of minimum consensus iterations T	
	$\mathcal{P}$	$\mathbf{z}$
4	18	29
6	23	25
8	22	35
10	17	15
20	15	38
50	21	29
100	17	26

We have to compute the volume and the RMSE for each agent because if we have multiple little networks instead of a single network, some agents will have $\mathcal{P}_i$ and $\mathbf{z}_i$ different from others. We only consider the contributions of agents that can communicate with at least one other agent. The noise standard deviation for each agent measurements was defined as $\sigma_i = 0.001||\mathbf{s}_i - \mathbf{p}||$, thus $\beta\hat{\sigma} = 0.001$, and the adjacency matrix was defined considering the communication range as 55% of the length of the diagonal of the square. This means that if two agents are at a distance lower than 55% of the length of the diagonal of the square of each other they communicate.

So the test was performed for different numbers of agents to try to obtain a value for T that could be generalized for all sizes of networks. The numbers of agents chosen were 4, 6, 8, 10, 20, 50 and 100, resulting in an average node degree of approximately 3, 4, 6, 7, 13, 39 and 80, respectively. For the first 5 cases we defined the length of the side of the space as 10 resulting in a grid with 100 possible positions. For the remaining two cases, the network becomes too big for this space so the length was changed to 100 and the grid now has 10000 possible positions. Only one target was considered in these experiments because, as seen in Section 4.1, the consensus is made independently for each target, and thus, adding more targets will only increase the number of consensus processes that we need to repeat but the number of consensus iterations stays the same.

We performed Monte Carlo runs which consist in repeating the same experiment, with random components, multiple times and from the different results it is possible to draw a probabilistic distribution. For each number of agents, 10 Monte Carlo runs with different deployment positions were made. After the last run, an average for the volume error vectors obtained for the different runs were made. Lastly, both resulting average volume error vector and RMSE were plotted and analyzed. These plots all start varying a lot for few consensus iterations but as the number of total consensus iteration increases the error starts to stabilize. So for each plot and for each number of agents the approximate number of consensus iterations for which the error reaches a plateau was registered through graphic analysis. This data is presented in Table 5.1.



(a) Difference of volumes.

(b) RMSE (Normalized).

Figure 5.1: Average of the curves obtained for the difference of volumes and RMSE for all the different number of agents. Both errors start at a high value but rapidly decrease, reaching a plateau in 10 iterations for **(a)** and 30 iterations for **(b)**. These plots are important not because of the presented values themselves but to show the behavior of the curves.

In Figure 5.1 we represent the average of all the curves obtained for all network sizes. This is a way to show the tendencies of the curves for each number of agents. Analyzing Table 5.1 we can conclude that a number of iterations around 20 is enough to reach the best approximations. So for the rest of the numerical result we will use $T = 20$.

We also test different efficient state-of-the-art distributed estimators to see how our method performs compared to them. The methods considered were the basic consensus method given by

$$\boldsymbol{\mathcal{P}}_i(\tau+1) = \sum_{j\in\mathcal{N}_i} \mathcal{W}_{ij}\boldsymbol{\mathcal{P}}_j(\tau) \tag{5.2}$$

$$\mathbf{z}_i(\tau+1) = \sum_{j\in\mathcal{N}_i} \mathcal{W}_{ij}\mathbf{z}_j(\tau), \tag{5.3}$$

with $\boldsymbol{\mathcal{P}}_i(0) = \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1}\mathbf{A}_i$ and $\mathbf{z}_i(0) = \mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} y_i(0)$, the traditional consensus + innovations method, given by

$$\boldsymbol{\mathcal{P}}_i(\tau+1) = \frac{\tau}{\tau+1}\sum_{j\in\mathcal{N}_i} \mathcal{W}_{ij}\boldsymbol{\mathcal{P}}_j(\tau) + \frac{1}{\tau+1}\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1}\mathbf{A}_i \tag{5.4}$$

$$\mathbf{z}_i(\tau+1) = \frac{\tau}{\tau+1}\sum_{j\in\mathcal{N}_i} \mathcal{W}_{ij}\mathbf{z}_j(\tau) + \frac{1}{\tau+1}\mathbf{A}_i^T C_{\mathbf{y}_{ii}}^{-1} y_i(\tau+1), \tag{5.5}$$

and a modified consensus + innovations method where the terms in the second sum in equations (3.56) and (3.57) were also multiplied by the corresponding weights.

The setup in this experiment was the same used before for the network with 100 agents but now the communication range was lowered, defined as 30% of the length of the diagonal of the square, resulting in an average node degree of 39, and only 1 run was made. We computed the error between

$\mathcal{P}$ obtained for each method and the Maximum Likelihood one for each agent of the network, given by $||\mathcal{P}_{\text{method},i} - \mathcal{P}_{ML}||$ and computed the empirical *Cumulative Distribution Function* (CDF).

According to [22], the CDF of a random variable at a certain value gives us the probability that this random variable will take a value less than or equal to the one we are analyzing. Translating this definition to the context of our problem this means that the value of the CDF for each method for a certain error gives us what is the probability of that method producing a matrix $\mathcal{P}$ with an error less than or equal to the one considered.

To better understand the results the error was normalized, which is obtained dividing the error by the norm of $\mathcal{P}_{ML}$. The same test was done for **z**.

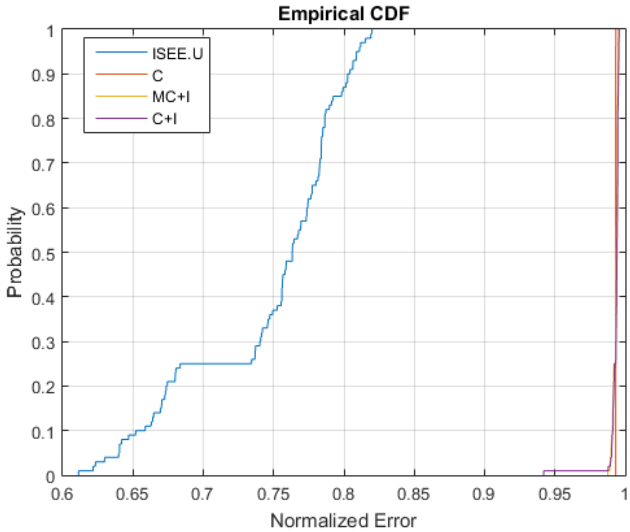


Figure 5.2: Empirical CDF of the normalized error for $\mathcal{P}$ for ISEE.U and multiple efficient estimators: consensus (C), consensus + innovations (C+I) and modified consensus + innovations (MC+I) for 20 consensus rounds. ISEE.U outperforms the other estimators.

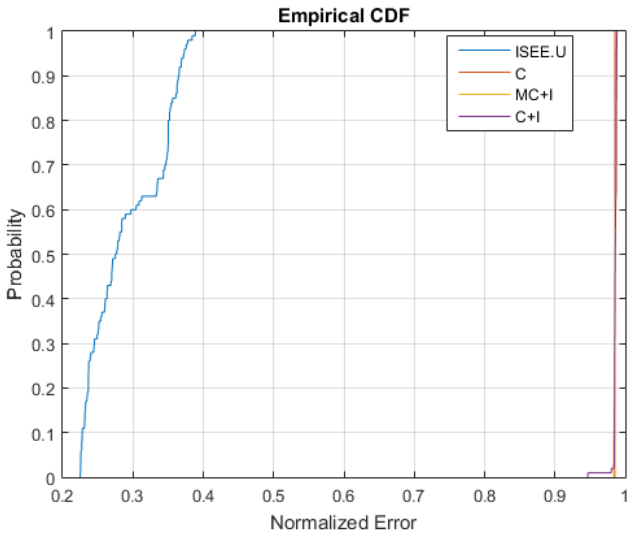


Figure 5.3: Empirical CDF of the normalized error for **z** for ISEE.U and multiple efficient estimators: consensus (C), consensus + innovations (C+I) and modified consensus + innovations (MC+I) for 20 consensus rounds. ISEE.U outperforms the other estimators.

From Figures 5.2 and 5.3 it is possible to conclude that ISEE.U clearly outperforms the other state-of-the-art estimators because it is much more probable for ISEE.U to obtain a much smaller normalized error than the other estimators. This results support the claims made in Section 3.3 where it was said that ISEE.U had a very clear advantage in mean error of the distributed quantities.

To validate ISEE.U we also decided to evaluate the behavior of the variance of the estimator with the number of consensus iterations τ made. To do so we considered a geometric network of 50 randomly deployed agents and one target presented in figure 5.4. The communications range of each sensor was manipulated so that the average node degree was 8.44. The noise standard deviation for each agent measurements was again defined as $\sigma_i = 0.001||\mathbf{s}_i - \mathbf{p}||$. 1000 Monte Carlo runs were made so that we could compute the covariance of **z** for one agent from which we also could compute the variance of the estimator.

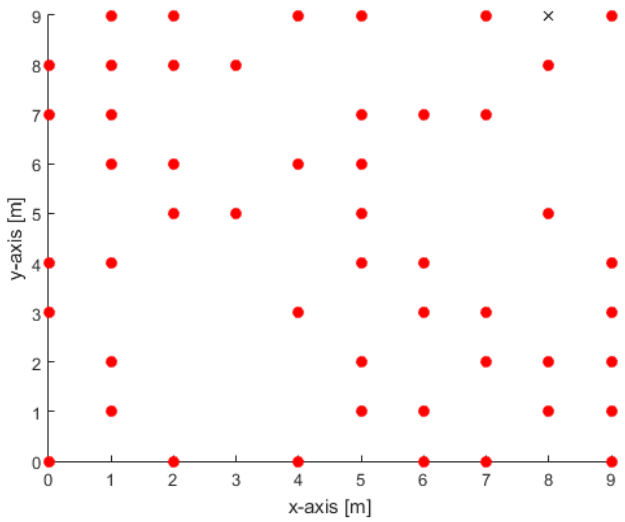
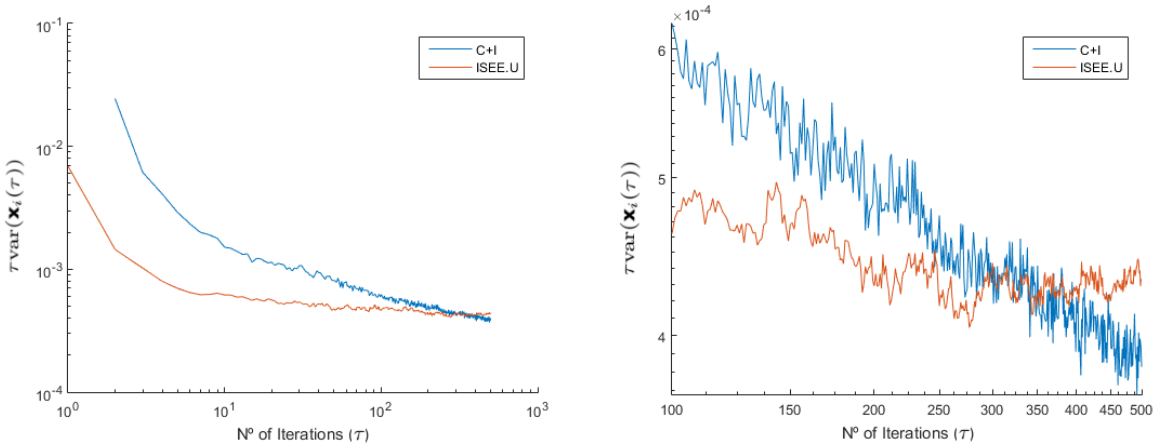


Figure 5.4: Setup used to compute the variance of the estimator.



(a) **(b)**

Figure 5.5: Variance of the estimator for ISEE.U and consensus + innovations (C+I). ISEE.U is very fast reaching a plateau in approximately 10 consensus iterations being surpassed by C+I after approximately 350 iterations. **(b)** is a zoom from the results presented in **(a)**.

From the Figures 5.5a and 5.5b it is possible to see that ISEE.U is faster than the traditional consensus + innovations method since ISEE.U reaches a plateau of low variance in roughly 10 consensus iterations while the traditional method needs approximately 350. Since the variance goes to 0 because more data is inputted during consensus we need to evaluate $\tau\mathrm{var}(\mathbf{x}_i(\tau))$ instead of only $\mathrm{var}(\mathbf{x}_i(\tau))$ to be able to evaluate the plateau of minimum variance. The results were expected because ISEE.U is not efficient, i.e., its variance does not touch the CRLB, unlike the vanilla consensus + innovations. Thus, the plateau of $\tau\mathrm{var}(\mathbf{x}_i(\tau))$ for consensus + innovations is indeed smaller, as depicted in Figure 5.5. In Figure 5.5b it is possible to see that the trade off is very small.

It is possible to conclude that, although not attaining the CRLB, ISEE.U is very fast and can reach variances very similar to the ones obtained by an efficient method.

5.2 Comparison between the two methods

In this section we will present a numerical comparison between the two methods to estimate the target position: ISEE.U, which is based in the linear estimator presented in Section 3.2.1, and the biased linear estimator presented in Section 3.2.2.

To test both methods we started with the setup presented in Figure 5.6a. The space used is a 10×10 square and the network is formed by 4 agents and their initial positions was chosen to be far away from the target so it was possible to see the error of the estimates for different quantities of noise since it depends on the distance between the agents and the target. We ran 100 iterations of our algorithm where, in each iteration, one agent is randomly chosen to wake up and through consensus choose which is its next best position and move (or not) to that chosen position. This means that the minimum number of iterations that the network needs to move is 4 and that the network can move up to 25 times. This is more than enough for the agents to reach their absolute best position given the size of the space. We performed each run of 100 iterations 20 times and computed the *Mean Absolute Error* (MAE) of $\hat{\mathbf{p}}$ given by

$$\mathrm{MAE}(t) = \frac{\sum_{j=1}^{M} ||\mathbf{p} - \hat{\mathbf{p}}_j(t)||}{M} \tag{5.6}$$

which gives us the navigation error and where $\hat{\mathbf{p}}_j(t)$ is the estimate of the target position for iteration t in run number j. The RMSE computes the squared root of the expected value of the squared norm of the difference between the estimate and the real position of the target which will eventually lead to the expected value of the squared norm of the estimation error. If the estimator is unbiased this will only result in its variance which after squared becomes its standard deviation. So for the unbiased estimator the RMSE gives us the standard deviation of the estimation error. For a biased estimator the difference between the estimate and the real position of the target will result in the estimation error plus the bias of that estimator so the RMSE will not just give the standard deviation of the estimation because it also has to consider the bias. On the other hand, the MAE is the expected value of the absolute value of the error of estimation which gives the average magnitude of the error for the unbiased estimator. For the biased estimator this quantity will be influenced by the absolute value of the bias. If the absolute value is not used the MAE becomes the mean bias error since the errors can cancel themselves and only the bias continues present.

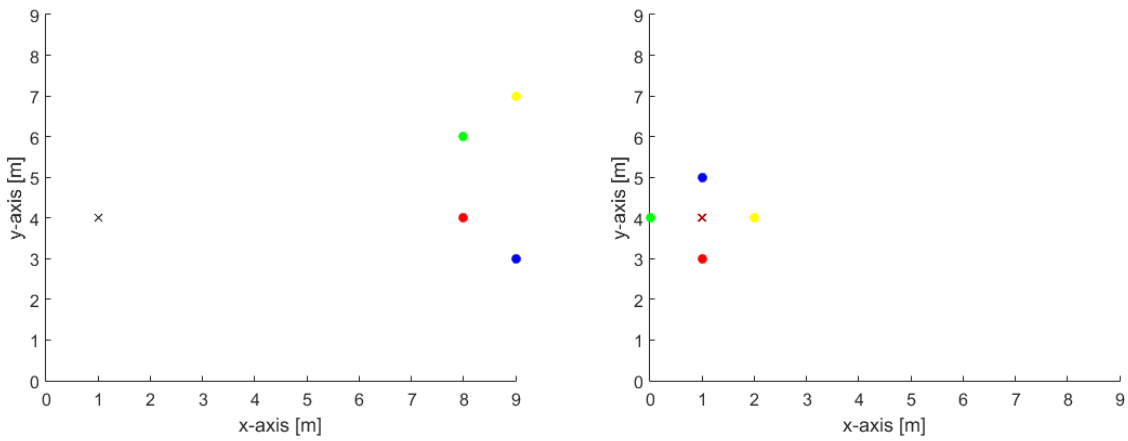
The RMSE was also computed but only for the last estimate of each run to see the variations that occurred in the RMSE as the number of runs is increased. So for this experiment the RMSE was given by

$$\mathrm{RMSE}_M = \sqrt{\frac{\sum_{j=1}^{M} ||\mathbf{p} - \hat{\mathbf{p}}_j(100)||^2}{M}}. \tag{5.7}$$

For the two quantities previously describe we just use the second and third position of $\hat{\mathbf{x}}$ because the first one is given by $||\mathbf{p}||^2$ and the dependency of it with the two other positions was neglected in 3.1 so we could solve this problem, and so, we will just compare the estimated position with the real one. The relation between $\hat{x}_1$ and $||[\hat{x}_1 \;\; \hat{x}_2]^T||^2$ will be later analyzed.

The range of each agent was again defined as 55% of the length of the diagonal of the space. The noise's standard deviation of each agent was changed between two factors so we could see its influence in both methods. The factors chosen were $\beta\hat{\sigma} = 0.001$ and $\beta\hat{\sigma} = 0.01$ of the distance between the agent and the target. In Figure 5.6b it is possible to see the final configuration of the network when the last iteration of the algorithm is reached, where this configuration represents the positions of the agents where their estimate of the target position is the most accurate.

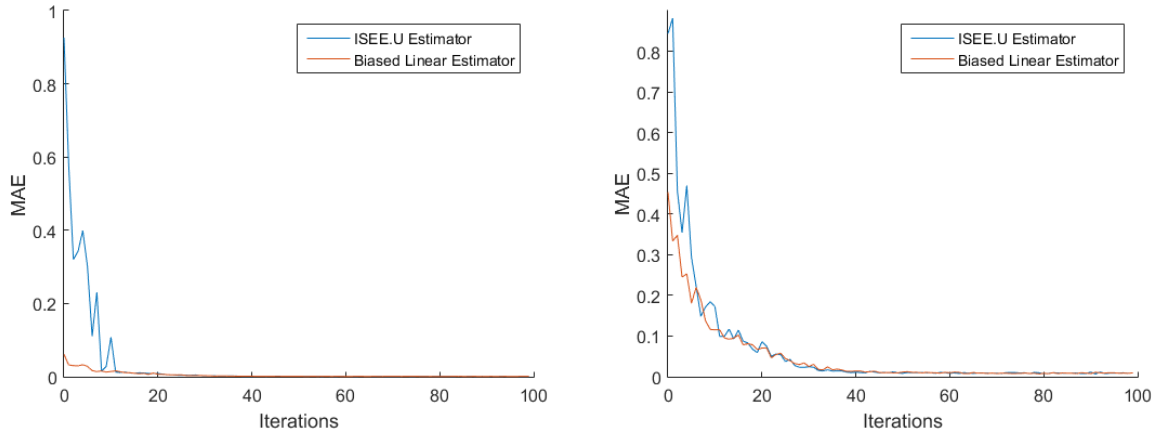
In each iteration a random agent wakes up and moves to the best position if it is not the one where it is. Then, after the consensus, it computes an estimate for the target position with the ISEE.U estimator and the biased linear estimator, using different data. Each estimate is then compared with the real position of the target.



(a) Initial setup. **(b)** Final setup.

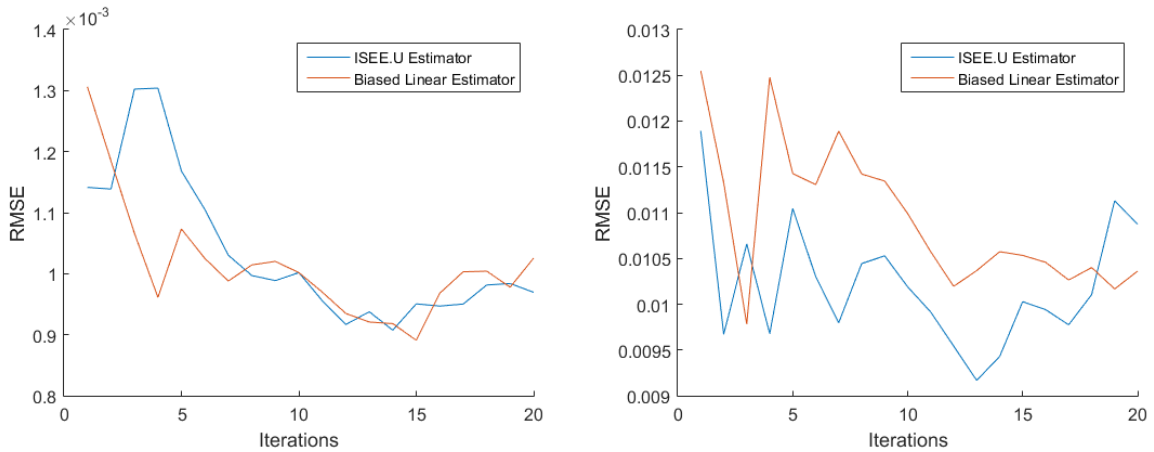
Figure 5.6: Initial and final setups for the comparison. The circles represent the agents, the black cross is the target and the red cross is the estimate for the target position.

In each run we also computed the bias for the biased linear estimator given by (3.44) in order to understand how the bias influences the results from that estimator.



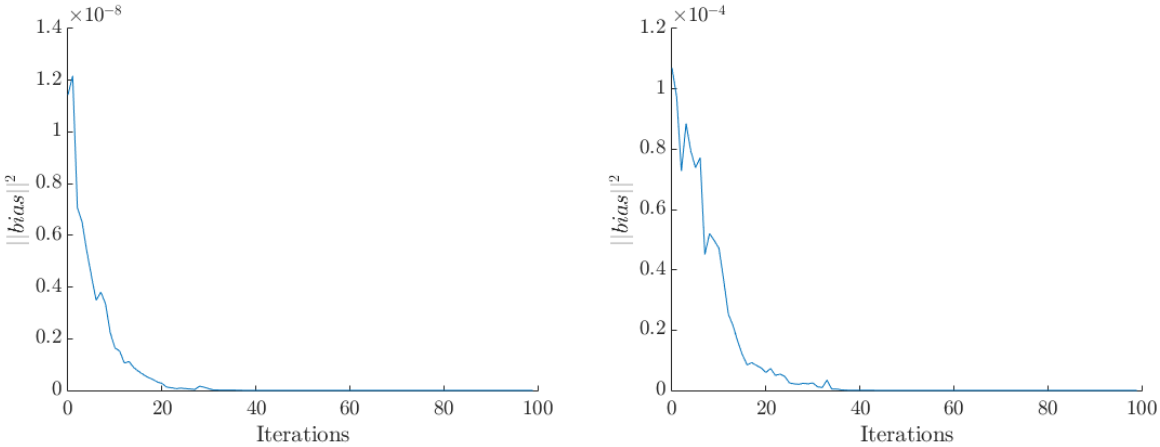
(a) $\beta\hat{\sigma} = 0.001.$ **(b)** $\beta\hat{\sigma} = 0.01.$

Figure 5.7: MAE for ISEE.U and for the biased linear estimator for different noise factors. The biased linear estimator outperforms ISEE.U in the first iterations. After 15 iterations in **(a)** and 40 iterations in **(b)** their performance is almost equal.



(a) $\beta\hat{\sigma} = 0.001.$ **(b)** $\beta\hat{\sigma} = 0.01.$

Figure 5.8: RMSE for ISEE.U and for the biased linear estimator for different noise factors. The covariance of the error is very small for both methods.



(a) $\beta\hat{\sigma} = 0.001$.

(b) $\beta\hat{\sigma} = 0.01$.

Figure 5.9: Squared norm of the bias for the biased linear estimator for different noise factors. The bias is very small when compared to the estimation error and increases with the noise.

It is possible to see in Figures 5.7 that the biased estimator outperforms the unbiased ISEE.U in every test. This can be explained by the fact that, although both estimators already consider an approximation of the problem since the dependency of the first position of the solution vector with the two other positions was neglected, ISEE.U considers also another approximation of the problem to remove the bias from the data model, assuming (3.19). After another approximation it is normal that the results turn out to be worse. As it is possible to see in Figures 5.9 the bias is very small when compared to the error so its influence in the estimation is very small, which still makes the biased linear estimator produce excellent estimates for the original linear model presented in equation (3.6).

Other explanation can be the fact that the movement of the agents is based on the covariance matrix of the estimator and the bias does not affect the covariance matrix that results from the biased linear estimator. This means that the difference between the covariance matrix of both estimators is only based on the covariance matrix of the measurements which is given by 3.26 for ISEE.U and by 3.38 for the biased linear estimator. Since the covariance matrix of the measurements used in the biased linear estimator is computed from the model without the extra approximation it describes it better than the other matrix, so the control of the agents with the biased linear estimator is better than with ISEE.U and this control can have influence on the future position estimates.

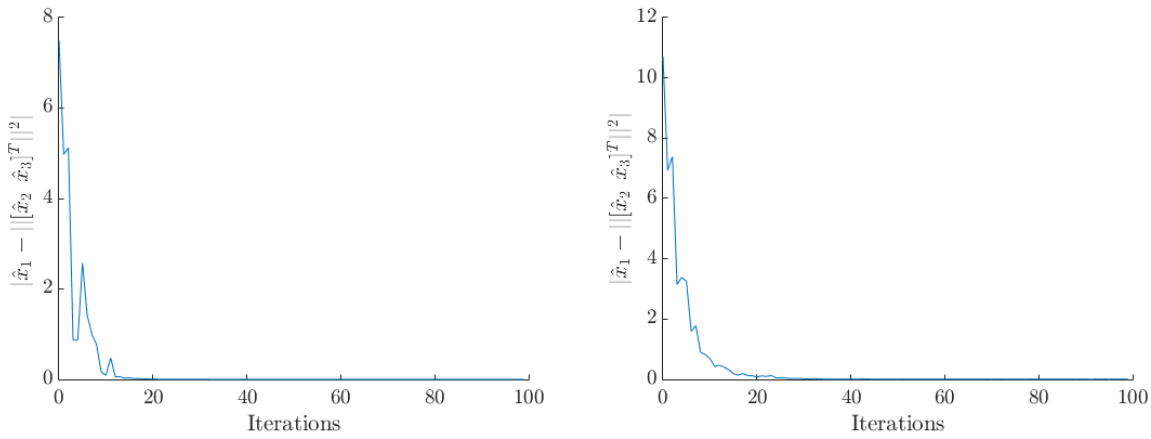
Finally comparing the bias for different noise factors in Figures 5.9a and 5.9b it is possible to see that the noise increase increases severely the bias by four orders of magnitude. This increase is also observable in Figure 5.7b when compared to 5.7a where the MAE for the biased linear estimator increases much more than the same quantities for ISEE.U. This can be due to the bias, given by (3.44) and $\mathbf{\Psi}$, defined by the variance of the noise and because the variance increases, the bias increases too.

Observing the figures in 5.8 we can see that the covariance of the error is very small for both methods and that performing more runs results in less oscillations in the RMSE which is also more noticeable when the noise is smaller.

In conclusion, we can say that both estimators produce good estimates for the target position but,

although being biased, the biased linear estimator outperforms ISEE.U. However, the biased linear estimator is also more influenced by noise so in some cases it may not provide the best estimates.

Now we will test the difference between the first component of the solutions vector and the squared norm of the others as it was discussed before. The settings of the test were the same as the previous test and the estimate was computed with ISEE.U.



(a) $\beta\hat{\sigma} = 0.001$. **(b)** $\beta\hat{\sigma} = 0.01$.

Figure 5.10: Relation between the first and the other two components of $\hat{\mathbf{x}}$ for different noise factors. The difference between the squared norms is considerable in the first iterations but it only takes 20 and 30 iterations in **(a)** and **(b)**, respectively, to reach a value close to 0.

In Figures 5.10 it is possible to see that the difference between both squared norms is considerable when the agents are far away from the target. When they start to approach it the difference becomes much smaller until eventually becoming close to zero. In the first iterations since the error is related with the distance between the network and the target the estimates have more error than in the following iterations so its normal that since the three components have different errors associated with them the squared norm of the latter two components does not match with the first. When the noise is increased the difference is accentuated but the behavior is very similar since this difference is also very considerable and as the iterations go by it eventually approaches zero. It is also important to say that for the case in 5.10b the difference needs several more iterations to become close to zero than in the case presented in 5.10a.

5.3 Refinement

Some estimates taken by the agents loose accuracy due to noise in the range measurements. The accuracy decreases with the increase of the distance between the agent network and the targets, as expected from everyday experience. To try to produce better estimates for all positions of the network we implemented a refinement phase after the target position estimation. This phase takes the current position estimate and tries to improve it. To do so, we use adaptations of two state-of-the-art methods,

presented in [30] and [31]. For each method, a general overview of it will be presented and the applied adaptations will be explained. Again, for the sake of simplicity we will only work with one target.

5.3.1 Distributed Gradient Algorithm With Barzilai-Borwein Stepsizes

The Distributed Gradient Algorithm With *Barzilai-Borwein* (BB) Stepsizes method was taken from [30] and it is a fully decentralized approach to solve the problem of autonomous distributed localization of agents in a network using range measurements to each other, with the use of a distributed gradient-based algorithm with BB stepsizes. Basically this method is implemented in agent networks where the nodes do not know their positions, so we give an initial estimate of the position of the nodes so that they can iterate through the gradient descent and through communications to anchors, that know their positions, and between agents, to reach the real configuration of the network. To adapt this theory to our case we have to imagine our target as an agent that wants to self locate and the other agents as anchors. So we now consider as communication graph $\mathcal{G}$, that was previously considered as the measurements graph, because only the target will "communicate" with the agents. An example of this graph can be seen in Figures 4.2a and 4.2b.

So this method starts by defining a cost function that we wish to minimize as

$$f_{R1}(\mathbf{x}) = \frac{1}{2} \sum_{i \in \mathcal{N}_{\mathbf{p}}} (||\mathbf{s}_i - \mathbf{p}||^2 - r_i^2)^2 \tag{5.8}$$

since the range is defined as

$$r_i^2 = ||\mathbf{s}_i - \mathbf{p}||^2 + w_i. \tag{5.9}$$

The first adaptation made to this method is to define the range as it was defined in (1.1). Consequently we have to change the considered cost function to

$$f_{R1}(\mathbf{x}) = \frac{1}{2} \sum_{i \in \mathcal{N}_{\mathbf{p}}} = (||\mathbf{s}_i - \mathbf{p}|| - r_i)^2 \tag{5.10}$$

where $\mathcal{N}_{\mathbf{p}}$ is the set of agents that belong to the neighborhood of the target that have range measurements relative to it. This new cost function is basically the one defined in (3.1).

The gradient descent [25] is used to find a local minimum using steps proportional to the negative of the gradient of the function, which is the same as moving our estimate through the direction that minimizes the function. This can be done using variable or fixed stepsizes. Since this method only guarantees to find a local minimum it is important to choose a good starting point for the iteration. This point will be given by the ISEE.U estimate. So in order to improve the position of the target we have to update according to

$$\mathbf{p}^{(t+1)} = \mathbf{p}^{(t)} - \alpha_t \nabla f_{R1}(p^{(t)}). \tag{5.11}$$

Barzilai and Borwein [32],[33] proposed a two-point sepsize for the gradient descent method that requires less computational effort and achieves better performance than the classical one. This stepsize

α_t is given by

$$\alpha_t = \frac{||\mathbf{p}^{(t)} - \mathbf{p}^{(t-1)}||^2}{\left(\mathbf{p}^{(t)} - \mathbf{p}^{(t-1)}\right)^T \left(\nabla f_{R1}(\mathbf{p}^{(t)}) - \nabla f_{R1}(\mathbf{p}^{(t-1)})\right)}. \tag{5.12}$$

This is the implementation of the gradient update for a centralized configuration. In [30] we can find a distributed approach to this gradient update. For our problem the target has to compute the stepsizes through consensus between the agents of the network and then it can iterate the estimation of its position through the gradient algorithm.

To perform the distributed computation of the stepsize two scalar values ρ and ψ are used to spread the information through the network. They are initialized as

$$\begin{aligned} \rho_m(t(0)) &= ||\mathbf{p}^{(t)} - \mathbf{p}^{(t-1)}||^2, \\ \psi_m(t(0)) &= \left(\mathbf{p}^{(t)} - \mathbf{p}^{(t-1)}\right)^T \times \left(\nabla_i f_{R1}(\mathbf{p}^{(t)}) - \nabla_i f_{R1}(\mathbf{p}i^{(t-1)})\right). \end{aligned} \tag{5.13}$$

These two scalars at iteration $\tau = 0$ are equal to 0 for the agents since they do not move and so $\mathbf{s}_i^{(t)} - \mathbf{s}_i^{(t-1)} = 0$. These scalars are then updated during the consensus iterations by

$$\begin{aligned} \rho_m(t(\tau+1)) &= W_{mm}\rho_m(t(\tau)) + \sum_{j \in \mathcal{N}_m} W_{mj}\rho_j(t(\tau)), \\ \psi_m(t(\tau+1)) &= W_{mm}\psi_m(t(\tau)) + \sum_{j \in \mathcal{N}_m} W_{mj}\psi_j(t(\tau)), \\ \tau &= 0, 1, \ldots T \end{aligned} \tag{5.14}$$

where $m \in \mathcal{V}$, $\mathcal{N}_m$ represents the neighborhood of m and **W** is another weight matrix that for our problem is an $(n+1) \times (n+1)$ matrix. Considering $|\mathcal{N}|$ as the cardinality of $\mathcal{N}$, the weight matrix can be filled as:

- If the agent can measure the target then the entry of its line relative to itself will be $1 - \frac{1}{|\mathcal{N}_{target}|}$ and for the target will be $\frac{1}{|\mathcal{N}_{target}|}$. The rest of the entries of the line will be 0;
- If the agent can not measure the target then his entire line will be 0;
- The line relative to the target has the value $\frac{1}{|\mathcal{N}_{target}|}$ in the entry relative to every agent with which it "communicates". The entry relative to itself will have the value 0.

The distributed stepsizes can be computed simply as

$$\alpha_{target}(t(\tau)) = \frac{\rho_{target}(t(\tau))}{\psi_{target}(t(\tau))}. \tag{5.15}$$

Finally the distributed gradient method yields that the estimate of the position of the target should be iterated according to (5.11), using $\alpha_{target}(t)$ and

$$\nabla f_{R1}(\mathbf{p}) = \sum_{i \in \mathcal{N}_\mathbf{p}} \left(\frac{\mathbf{s}_i - \mathbf{p}}{||\mathbf{s}_i - \mathbf{p}||}\right)\left(||\mathbf{s}_i - \mathbf{p}|| - r_i\right), \tag{5.16}$$

is the gradient for the new cost function defined as (5.10).

The adaptations made on the algorithm presented in [30] were only in the gradient update since we only consider an active node (target). It was also verified that only one warm-up iteration was enough to correctly start the algorithm. Although the presented algorithm considers two stopping criteria, the implemented algorithm only stops when the desired gradient iterations are completed. The stopping criterion that stops the algorithm if the estimates for consecutive iterations are almost equal was removed because it can stop the program in a plateau of the cost function f and not in the local minimum and so the position estimate would not be accurate.

For this method, the information that is broadcast between the agents and the target is composed by the positions of the agents, the range measurements, and, during the consensus, each agent broadcasts its ρ and ψ to the target and the target broadcasts the same quantities to the agents. So the number of communications for this method is given by

$$\#\text{communications} = 2TT_G(n+1) + n(d+1), \tag{5.17}$$

where T is again the total number of consensus iterations and T_G the total number of iterations for the gradient method.

5.3.2 Distributed ML agent network localization

The method presented in [31] is used to solve the problem of agent network localization by directly optimizing the nonconvex maximum likelihood criterion with a distributed way. Basically it considers a network with anchors, which know their positions, and agents, which do not know their positions and want to find it. So the agents through range measurements to the anchors and to other agents can use this distributed algorithm to find their positions and ultimately we can find the correct positions of all the agents in the network.

Mathematically, this is the same as trying to solve the optimization problem defined by the minimization of the cost function $f_{R2}(\mathbf{s})$, relative to $\mathbf{s}$, which is given by

$$f_{R2}(\mathbf{s}) = \sum_{i \backsim j} \frac{1}{2}(||\mathbf{s}_i - \mathbf{s}_j|| - d_{ij})^2 + \sum_i \sum_{k \in \mathcal{A}_i} \frac{1}{2}(||\mathbf{s}_i - \mathbf{a}_k|| - r_{ik})^2, \tag{5.18}$$

where d_{ij} is the range between agents i and j, $\mathcal{A}_i$ is the set of anchors that have range measurements relative to agent i and $\mathbf{a}_k$ represents the position of anchor k.

Again, we consider the target as an agent that is trying to locate himself and the n agents of the network as anchors in this method and hence this configuration is described as the measurements graph $\mathcal{G}$. So the cost function now takes the form of

$$f_{R2}(\mathbf{p}) = \sum_{i=1}^{n} \frac{1}{2}(||\mathbf{p} - \mathbf{s}_i|| - r_i)^2. \tag{5.19}$$

This problem can be also represented in the matrix form as

$$\min_{\mathbf{z}} f_{R2}(\mathbf{z}) = \frac{1}{2}\mathbf{z}^T\mathbf{M}\mathbf{z} - \mathbf{b}^T\mathbf{z} \tag{5.20}$$

$$\text{subject to } \mathbf{z} \in \mathcal{Z}, \tag{5.21}$$

where

$$\begin{aligned}
\mathbf{z} &= \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\omega} \end{bmatrix}, \mathbf{M} = \begin{bmatrix} \mathbf{E}^T\mathbf{E} & -\mathbf{E}^T \\ -\mathbf{E} & \mathbf{I}_{(nd)\times(nd)} \end{bmatrix}, \\
\mathbf{b} &= \begin{bmatrix} \mathbf{E}^T \\ -\mathbf{I}_{(nd)\times(nd)} \end{bmatrix} \boldsymbol{\alpha}, \quad \boldsymbol{\alpha} = \begin{bmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_n \end{bmatrix}, \\
\mathbf{E} &= \begin{bmatrix} \mathbf{I}_{d\times d} \\ \vdots \\ \mathbf{I}_{d\times d} \end{bmatrix},
\end{aligned} \tag{5.22}$$

$\mathcal{Z} = \{\mathbf{z} = (\mathbf{p}, \boldsymbol{\omega}) : \omega_i = ||r_i||, i \in V\}$ and where $\mathbf{I}$ is an identity matrix where the subscript represents its sizes.

The solution to (5.20) is given by the projected gradient method [34] where

$$\mathbf{z}^{t+1} = P_{\mathcal{Z}}\left(\mathbf{z}^t - \frac{1}{L}\nabla f(\mathbf{z}^t)\right), \tag{5.23}$$

$$\nabla f_{R2}(\mathbf{z}) = \mathbf{M}\mathbf{z} - \mathbf{b}, \tag{5.24}$$

is the projected gradient iteration, where $P_{\mathcal{Z}}(x)$ is the projection of point x onto $\mathcal{Z}$ and L is the Lipschitz constant, that can be obtained by

$$L = \lambda_{max}(\mathbf{E}\mathbf{E}^T) + 1. \tag{5.25}$$

Basically the algorithm starts with an estimate of the position of the target $\mathbf{p}_{\text{est}}$ that is given by ISEE.U. Then, for each iteration it computes the gradient of the cost function, computes the new $\mathbf{z}^{t+1}$ using the gradient method, with a fixed stepsize given by the inverse of the Lipschitz constant, and finally it applies the projection in the $\boldsymbol{\omega}$ component of $\mathbf{z}^{t+1}$. After the final iteration, the estimate for the position of the target will be given by the first d components of $\mathbf{z}$. In [31] it is only considered that the anchors only broadcast their positions to the agents that are trying to locate themselves. Since the target cannot make range measurements we consider that each agent broadcasts its own position and the range measurement thus the number of communications is given by

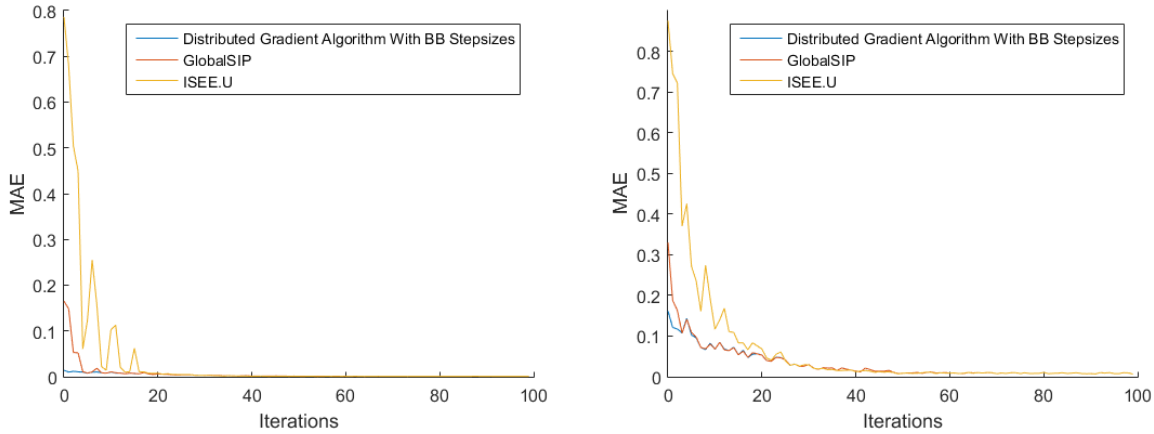
$$\#\text{communications} = n(d+1). \tag{5.26}$$

5.3.3 Numerical experiments

Now we will put the two presented refinement methods to test in order to draw some comparisons between them and the estimates produced by the ISEE.U estimator.

To do so, we started again with the setup presented in Figure 5.6a. The space used is a 10×10 square and the network is formed by 4 agents and their initial position was chosen to be far away from the target so we could see the influence of the refinement methods in the estimates at various distances from it. Again, 100 iterations were ran 20 times and the range of each target was again defined as 55% of the length of the diagonal of the space. The noise's standard deviation of each agent was changed between two factors so we could see the influence of the refinement methods on different noisy environments. The factors chosen were $\beta\hat{\sigma} = 0.001$ and $\beta\hat{\sigma} = 0.01$ of the distance between the agent and the target. For the refinement methods, we computed 100 iterations of the gradient method for each one and for the Distributed Gradient Algorithm With BB Stepsizes we computed 1 warm up iteration and 20 consensus iterations. Again, in Figure 5.6b it is possible to see the final configuration of the network when the last iteration of the algorithm is reached, where this configuration represents the positions of the agents where their estimate of the target position is the most accurate.

The experiment consists in measuring the accuracy of the estimates of the target position relative to the real one. To do so, in each iteration the randomly selected agent moves to the best position and through consensus computes an estimate using the distributed ISEE.U estimator. This estimate is then used as initial estimate for the refinement methods. We then compute the MAE for each of the three computed estimates as in (5.6).



(a) $\beta\hat{\sigma} = 0.001$. **(b)** $\beta\hat{\sigma} = 0.01$.

Figure 5.11: MAE for ISEE.U and for the two refinement methods presented in this section for different noise standard deviation factors. The refinement methods outperform ISEE.U in the first iterations but the three methods start producing identical estimates after 20 iterations.

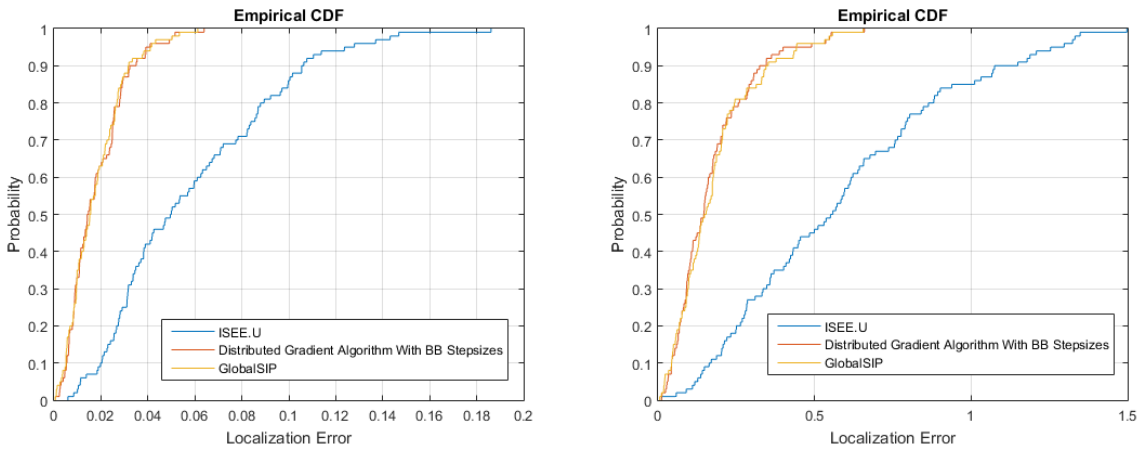
The results of this experiment can be observed in Figures 5.11. In both figures it is possible to see that the refinement process improves the estimation mostly in the first iterations, where the agents are further away from the target and the measures have a bigger noise component. After approximately 20 iterations, the network is already close enough to the target to compute very accurate estimates through the ISEE.U estimator, since the three methods produce almost equal results. You can also notice that for some iterations the refinement even produces worst estimates than the initial one. The cause for this phenomenon is the fact that the cost function is parametrized by the actual noisy range

measurements, and the shape and extreme of the ML function will change with different instances of these measurements. Thus, the global minimum of the ML cost will not coincide with the true positions, but will be at a short distance from them. If our initialization is very good we can even see the localization error growing when the cost is declining.

Comparing now Figures 5.11a and 5.11b we can see, as expected, that noisier measures increase the error of estimation. So it is normal that the ISEE.U fares worse in the first iterations for the noisier case presented in 5.11b but as it was seen in Section 5.2 the approximation made so that our proposed ISEE.U estimator could be used is robust to noise, that does not affect the estimate that much. The refinement methods also provide worse estimates with increasing noise but as we can see the Distributed Gradient Algorithm With BB Stepsizes method is more prone to noise than the Distributed ML agent network localization (GlobalSIP) method which is noticeable because the GlobalSIP reaches the same MAE as the other refinement method much earlier than in the case where the noise was smaller.

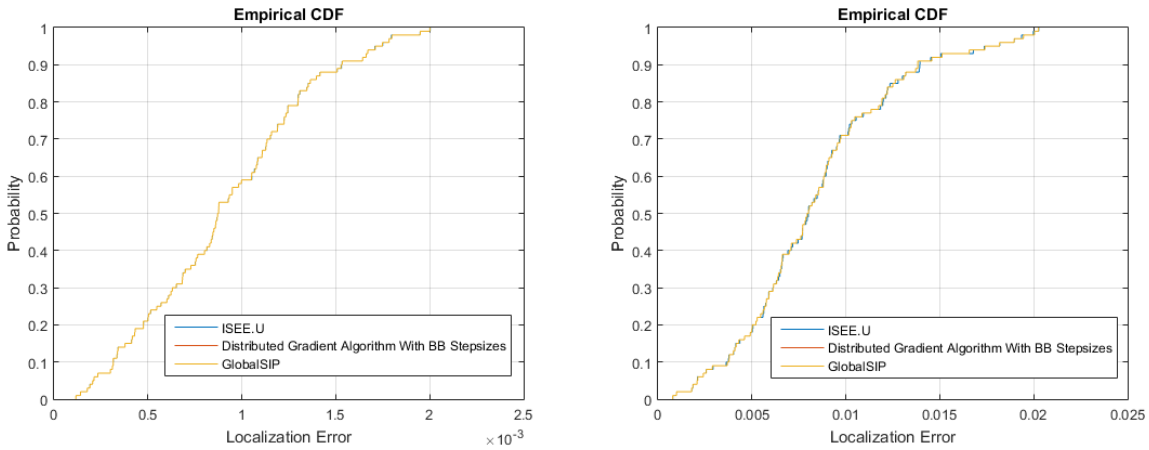
Finally, it is also important to compare the behavior of the two refinement methods. In Figures 5.11 we can see that initially the refinement methods produce different errors from each other being the GlobalSIP the one that produces the worst results. After some iterations it becomes very hard to distinguish the behavior of the two since they produce almost equal estimates. The difference between the MAE of the two methods may be related to the number of communications made between the agents and the target in each method. From equation (5.17) and the data presented above we can say that the Distributed Gradient Algorithm With BB Stepsizes method does 20012 communications and from equation (5.26) it is possible to see that the GlobalSIP method only does 12, which is a huge difference in the number of communications.

Another way to analyze these results is through the CDF. The experiment consisted in, for the initial and final setup, running 100 Monte Carlo iterations where a random agent awakes and, through the distributed ISEE.U estimator, estimates target position that is after used by the refinement methods to produce their target position estimates. Then we compute the localization error given by $||\mathbf{p} - \hat{\mathbf{p}}||$ for each estimate considering the real position of the target. These two setups were chosen to draw comparisons between the methods when the network is far away from the target and when it is very close, corresponding to the first and last iterations in Figures 5.11, respectively. For each setup we again also test different noise factors.



(a) $\beta\hat{\sigma} = 0.001$. **(b)** $\beta\hat{\sigma} = 0.01$.

Figure 5.12: Empirical CDF of the RMSE for ISEE.U and for the two refinement methods considering the initial setup for different noise standard deviation factors. The refinement methods clearly outperform ISEE.U.



(a) $\beta\hat{\sigma} = 0.001$. **(b)** $\beta\hat{\sigma} = 0.01$.

Figure 5.13: Empirical CDF of the RMSE for ISEE.U and for the two refinement methods considering the final setup for different noise standard deviation factors. The refinement methods and ISEE.U have almost equal performances.

The obtained CDFs support the conclusions drawn before. For the estimates made for the initial setup in 5.12 it is possible to see that the refinement improves significantly the ISEE.U estimate since it is more probable for the estimates to have a lower error after the refinement than the original one. The increase in noise maintains the same relation between the curves.

The CDFs obtained for the final setup presented in 5.13 show that when the agents are close enough to the target the ISEE.U estimate is accurate enough and the refinement does not improve it significantly. In some parts of the curves the ISEE.U estimate even provides better results. With the increase of noise, the error increases as well but the tendency of the CDF stays the same.

The two refinement methods have similar performances for the initial setup where it is possible to see that the Distributed Gradient Algorithm With BB Stepsizes method is just slightly better than GlobalSIP.

In conclusion the refinement methods are useful when the network is far away from the target and its

measurements have a lot of noise. When the network approaches the ideal configuration the refinement can be turned off because the ISEE.U estimates are as good or sometimes even better that the ones after the refinement.

Between the refinement methods, they presented very similar results through the experiments to which they where subjected, with the Distributed Gradient Algorithm With BB Stepsizes method slightly outperforming the GlobalSIP method in the first iterations. As it can be observed above, the Distributed Gradient Algorithm With BB Stepsizes method uses a lot more information and so has to make a lot more communications than the GlobalSIP method. For some setups the number of communications can be critical because, for example, the agent's battery can be exhausted, which can lead to loose a node in the network. Also, since the Distributed Gradient Algorithm With BB Stepsizes method has to do consensus iterations inside the gradient method iterations one can easily see that this method will take a lot more computational time than the GlobalSIP method.

Although the refinement process is very useful to improve our estimate in certain cases, the two presented methods cannot be implemented on our method directly. This is because the refinement methods assume that the target is a cooperative agent, that can communicate and exchange information with agents, and it can make computations since these methods assume that the target tries to locate itself. For those reasons the refinement methods cannot be implemented and can only be used to validate the estimate that comes from the ISEE.U estimator. Since we concluded that the refinement methods were only useful for certain positions of the network we can say that our estimator already provides accurate estimates.

5.4 Simulations with a state-of-the-art method

In this section we will describe some simulations made with both ISEE.U and the benchmark [3], already presented in Section 2.2, in different scenarios in order to draw some comparisons between the two and to numerically validate the method presented in this work against one state-of-the-art method.

We will start by defining the different parameters that were used in all of the following simulations. The space that was considered was a 200×200 square and thus our grid has now 40000 possible positions for the agents. We had to consider a larger space than the one used in the previous experiments because the benchmark method uses samples to represent the belief of the target position and a small grid could not represent a big number of samples. The standard deviation for the noise continues to depend on the distance and was defined as $\beta\hat{\sigma} = 0.1$ and applied to both methods which implied some changes in the benchmark since it only considers a single constant standard deviation for all agents or a standard deviation that used a model slightly different from ours that considered a pathloss factor.

For the ISEE.U method we continued to apply a communication range to each agent where its radius is again 55% of the length of the diagonal of the space. For the comparison with the benchmark the motion of the network in this method was also changed. Previously, in each iteration one agent randomly woke up and moved to the best position. Now in each iteration of the algorithm every agent wakes up and moves sequentially, so in every iteration of the simulation all the agents of the network have the

opportunity to move. After the movement of each agent an estimate for the target position for that agent is computed through consensus.

For the benchmark and following the Simulations Section presented in the corresponding paper, $120,000$ samples were used in the estimation layer and $6,000$ samples in the control layer.

For a defined number of iterations we ran both algorithms at the same time, and each algorithm produced an estimate for the target position and the control for all agents of the network for each run. This process was repeated 5 times so that the target estimate from both methods could be used to compute the MAE of estimation, given by

$$\mathrm{MAE}(t) = \frac{\sum_{j=1}^{M} ||\mathbf{p} - \hat{\mathbf{p}}_j(t)||}{M}, \tag{5.27}$$

for each method.

Since in ISEE.U each agent uses consensus to compute the estimate for the target position, every agent also has the same estimate, assuming that the information from one agent is able to reach all the network, and so we used the estimate obtained by a random agent in each iteration to compute the MAE.

The first simulation consisted in the usual network of four agents and one static target. The agents were deployed in random positions, resulting in the initial setup presented in Figure 5.14. The simulation ran for 150 iterations since this was enough to analyze the trajectories that result from both methods and the evolution of the estimation error.

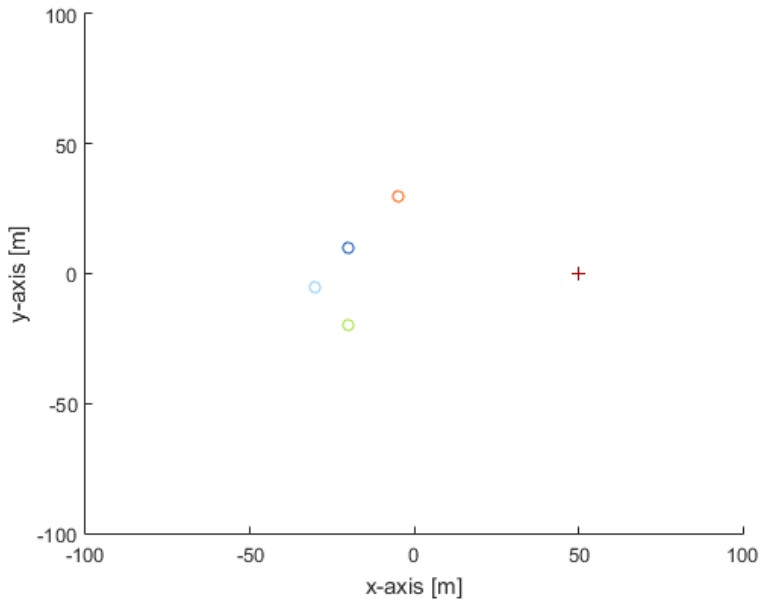
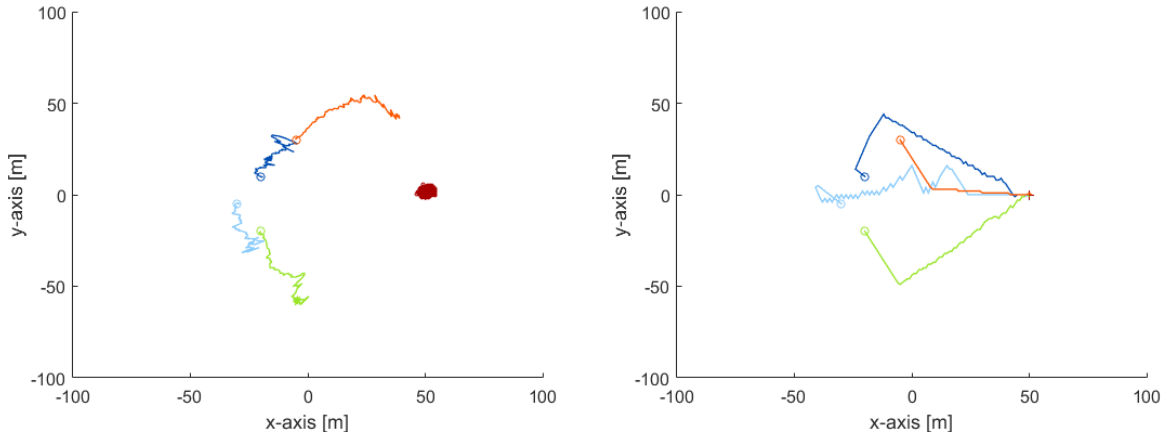


Figure 5.14: Initial setup for the first three simulations. The circles represent the agents positions and the red cross is the target position.



(a) Benchmark. **(b)** ISEE.U.

Figure 5.15: Obtained trajectories for the first simulation for both methods. The trajectories are represented by lines having the same color of the circle of the agent that they correspond. The red dots are the samples that represent the belief for the target position in the benchmark method. In **(a)** the agents are spreading at the same time they are going in the direction of the target while in **(b)** the agent move directly to the target.

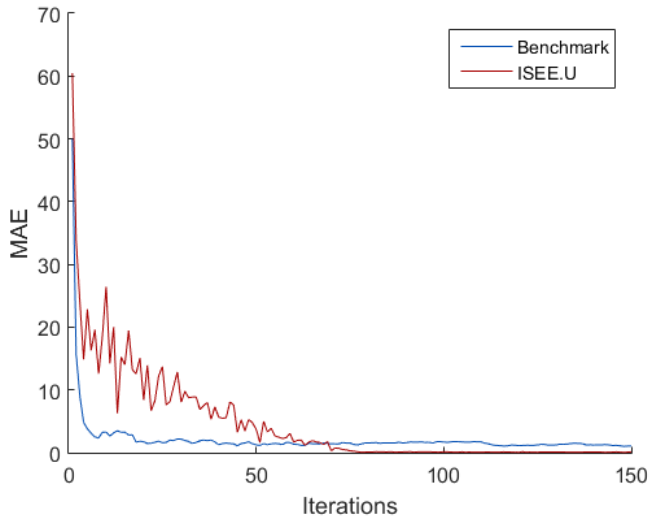


Figure 5.16: MAE obtained for both methods in the first simulation. ISEE.U outperforms the benchmark after approximately 75 iterations.

In the Figures 5.15 it is possible to observe the differences between the trajectories obtained for both methods, which were different because both methods used different cost functions to compute the respective control.

For the benchmark, we can see in Figure 5.15a that the agents start approaching the target and when they reach a distance from which it is not possible to decrease the measurement noise, the agents spread out to find a formation more suitable to locate and track the target.

With ISEE.U, as seen in Figure 5.15b the agents move in the direction of the target as it already happened in the previous tests throughout this chapter. This is because the cost function used to

compute the control is the volume of the error ellipsoid which is a function of the covariance matrix of the estimator from equation (3.29), dependent on the distance between the agents and the target via the covariance of the measurement data represented in equation (3.27). As the error is dependent on the distance, agents try to minimize their distance to the target.

From Figure 5.16 we can see the evolution of the MAE for both methods. We can see that, after approximately 10 iterations, the benchmark reaches a low MAE and stays around the same value for the rest of the run. Meanwhile ISEE.U needs about 70 iterations to reach the same value as the benchmark but the MAE for this method continues decreasing until approximately zero. This happens because without much movement the benchmark is able to reach a very good MAE and since the agents start to spread this value is not decreased throughout the run. In ISEE.U, the agents keep minimizing the distance between the network and the target which decreases the MAE until it is almost zero which happens when the agents reach the optimal positions which are the ones where the cost function is minimal.

For the second simulation we wanted to compare the performance of our method when tracking a mobile target. To do so, the same setup presented in Figure 5.14 was used but now the target state at some iteration t, $\mathbf{x}_{\text{target}}(t) = [p_1(t)\ \ p_2(t)\ \ \dot{p}_1(t)\ \ \dot{p}_2(t)]^T$ changes according to

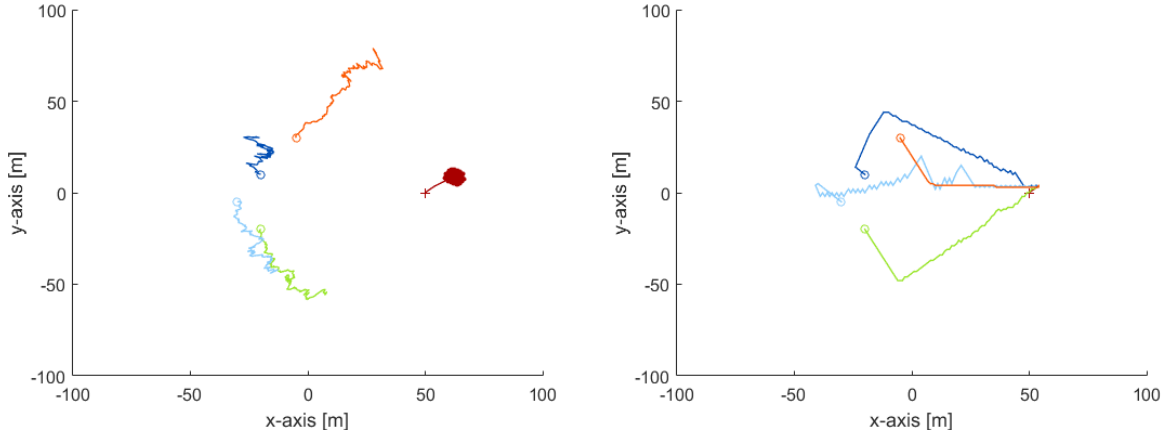
$$\mathbf{x}_{target}(t) = \mathbf{\Gamma}\mathbf{x}_{\text{target}}(t-1) + \mathbf{\Lambda}\mathbf{q}(t), \tag{5.28}$$

where

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{\Lambda} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{5.29}$$

and $\mathbf{q}(t) \in \mathbb{R}^2$ where $q_i(t) \sim \mathcal{N}(0, \tilde{\sigma}_q^2)$ is statistically characterized by a Gaussian distribution with zero mean and variance $\tilde{\sigma}_q^2 = 10^{-5}$. This is the same movement that the authors of the paper [3] gave to the target when testing the benchmark with a mobile target.

Since the target is now moving, the real target position changes with time, so for the computation of the MAE presented in equation 5.27 we now use $\mathbf{p}(t)$.



(a) Benchmark. **(b)** ISEE.U.

Figure 5.17: Obtained trajectories for the mobile target localization for both methods. The trajectories are represented by lines having the same color of the circle of the agent that they correspond. The red line represents the trajectory of the target. The red dots are the samples that represent the belief for the target position in the benchmark method. The agents in **(a)** spread more than in the previous simulation to cope with the movement of the target while the agents in **(b)** pursue the target and trap it.

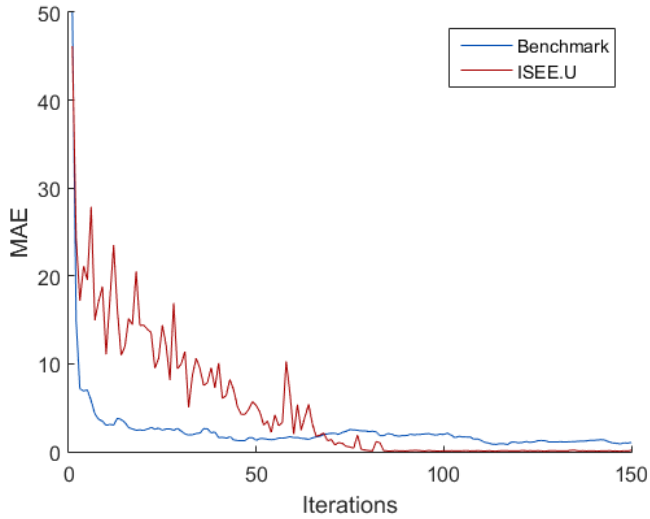


Figure 5.18: MAE obtained for both methods for the mobile target localization. ISEE.U outperforms the benchmark after approximately 70 iterations.

In this simulation it is possible to see in Figure 5.17a that the trajectories of the agents when using the benchmark method do not change that much comparing to the previous simulation. However when using ISEE.U, in Figure 5.17b we can see that the agents pursue the target and eventually end up trapping it.

For the MAE error in Figure 5.18 it is possible to see that the trend from the previous simulation continues since the benchmark reaches a good value with few iterations but is surpassed by ISEE.U after approximately 70 iterations. When the agents eventually trap the target the MAE decreases substantially

since we have a situation like the previous simulation where the target is static and the agents are in the optimal positions.

In both methods each agent moves at a maximum velocity of 1 position per iteration which means that the group also moves at this velocity. So the group velocity is 20 times greater than the target velocity so it is normal that the group quickly traps the target in ISEE.U.

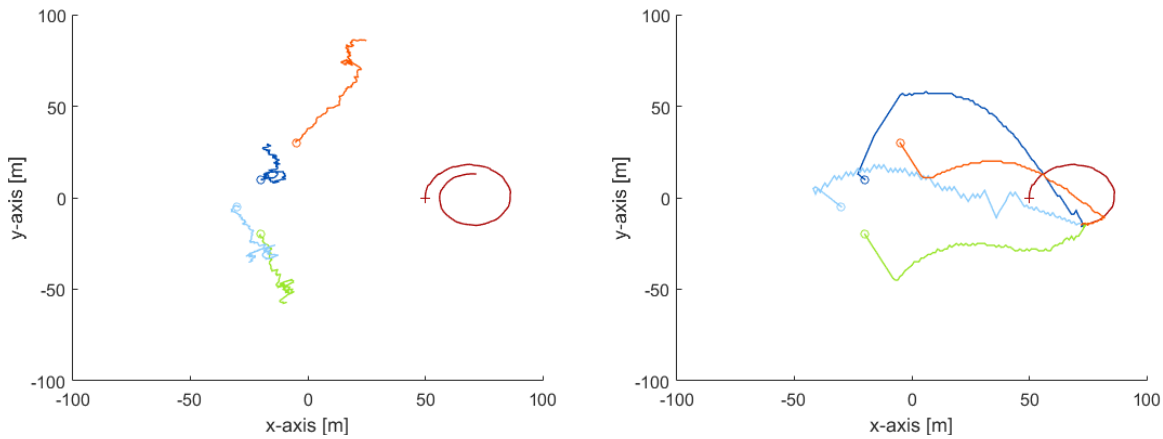
To challenge both algorithms in the third simulation we changed the movement of the target so it would move in a spiral trajectory, where the radius would decrease after some number of iterations, and we also increased the velocity of the target. So again we used the same setup from Figure 5.14 but now the evolution of the target state is given by

$$\mathbf{x}_{\text{target}}(t) = \mathbf{x}_{\text{target}}(t-1) + \mathbf{\Gamma}(t) + \mathbf{\Lambda}\mathbf{q}(t), \tag{5.30}$$

where $\mathbf{\Gamma}(t)$ is

$$\mathbf{\Gamma}(t) = \begin{bmatrix} R_t(t) * cos(\pi - \frac{v}{R(t)} * t) + c_1 \\ R_t(t) * sin(\pi - \frac{v}{R} * t) + c_2 \\ 0 \\ 0 \end{bmatrix}, \tag{5.31}$$

and where v is the linear velocity of the target that was kept constant and defined as 70% of the velocity of the group, $R_t(t)$ is the radius of the spiral that was decreased every 10 iterations according to $R_t(t) = 0.97R_t(t-1)$, where we defined $R_t(0) = 20$, and $\mathbf{c} = [c_1 \;\; c_2]^T$ are constants that define the center of the spiral, since the coordinates of the center of the spiral are given by $(c_1 + p_1(0), c_2 + p_2(0))$, and chosen as $\mathbf{c} = [20 \;\; 0]^T$. So the target will start at its normal starting position and will move clockwise in a spiral trajectory centered in $(70, 0)$.



(a) Benchmark. **(b)** ISEE.U.

Figure 5.19: Obtained trajectories for the spiral trajectory for both methods. The trajectories are represented by lines having the same color of the circle of the agent that they correspond. The red line represents the trajectory of the target. The agents in **(a)** spread even more than in the previous simulation since they loose track of the target as it can be seen by the absence of samples, while in **(b)** the agents once again pursue and trap the target.

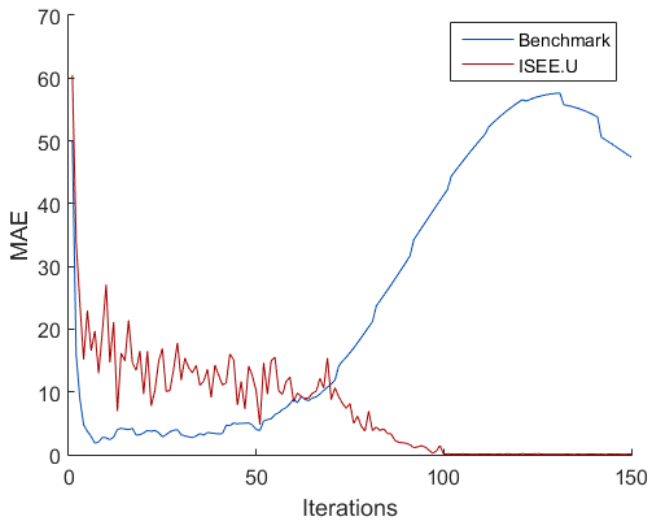


Figure 5.20: MAE obtained for both methods in the spiral trajectory. ISEE.U outperforms the benchmark after approximately 60 iterations. After 50 iterations the MAE for the benchmark increases.

For the spiral trajectory simulation it is possible to observe in Figure 5.19a that the trajectories of the agents change when compared to the other two simulations since the agents do not try to move towards the target as in the other simulations. Another difference from the other simulations is the fact that the samples that represent the belief for the estimate are not present. This is due to the fact that when the target is moving during the first semi-circle the benchmark assumes that the target has a linear trajectory so when its movement abruptly changes the agents still assume that the target is moving in a linear trajectory and continue to produce estimates until reaching the boundary of the space. This also can explain the fact that the agents start to move in the direction of the target but since they loose it and start computing wrong estimates, they start spreading to try to minimize that error.

For ISEE.U in Figure 5.19b the behavior is similar to the other simulations since the agents approach the target and eventually end up trapping it.

For the MAE in Figure 5.20 the behavior for ISEE.U is similar but for the benchmark method it changes a lot. As one can see the MAE for the benchmark still reaches a good value in few iterations but after approximately 50 iterations the MAE starts increasing. This coincides with the moment when the movement of the target abruptly changes and so the estimates start to be further away from the real target position and so the estimation error increases.

We wanted also to test what would be the influence on the trajectories and MAE of the ISEE.U method if the agents do not trap the target, which would be a behavior more similar to the benchmark method. To imprint this behavior on our method we defined that the agents could not occupy positions in which the measured range can not be smaller than a certain value. This creates a "safety" circle around the target inside which the agents can not enter and so the movement of the target is not interrupted. So the agents will continue to move normally, trying to minimize the localization error but at the same they evaluate if the best position to which they want to move is or is not inside the circle.

We repeated the same simulation as before but now considering four different radii for the circle

around the target. These different radii were defined as 5, 10, 20 and 50. In Figure 5.21 is is possible to observe the trajectory of the agents when the radius was defined as 50. It is possible to see that the agents initially have the same behavior like the one presented in Figure 5.19b but when they are approximately 50 units apart from the target they start to spread, which was the behavior showed by the agents for the benchmark method, and do not trap the target which continues its movement until the end of the run.

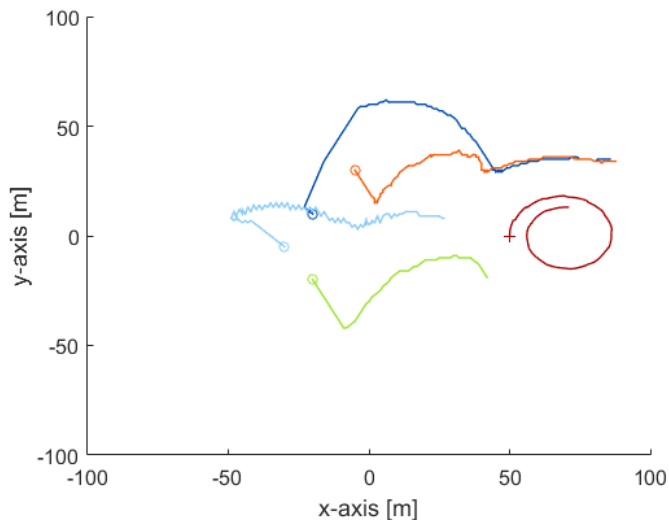


Figure 5.21: Obtained trajectories for the ISEE.U method considering a safety radius $R = 50$. The trajectories are represented by lines having the same color of the circle of the agent that they correspond. The red line represents the trajectory of the target. The existence of a safety radius makes the agents spread, showing a more similar behavior to the benchmark.

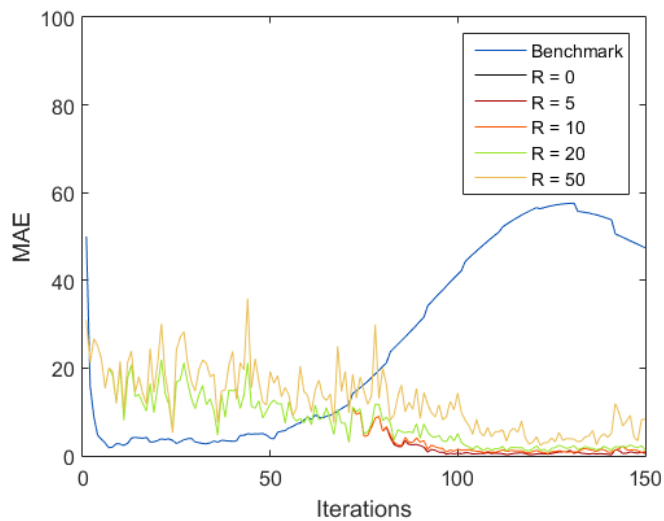


Figure 5.22: MAE obtained for the benchmark method and for the ISEE.U method considering different safety radii. The increase in the safety radius also increases the MAE but ISEE.U still outperforms the benchmark.

In Figure 5.22 a comparison between the previously obtained MAEs and the MAEs obtained consid-

ering the different radii is presented. As one can see the MAE for when there is no safety circle and for safety circles with small radii (5 and 10) is very similar and differences only show up after approximately 70 iterations since that is the time that the agents need to get close to the target and only then the influence of the circle starts changing their trajectories. For the bigger radii (20 and 50) the differences are more visible since the agents do not have to move a lot to have their trajectories changed by the safety circle. Although safety circles with big radii keep the agents far away from the target, the network never looses track of it and is still able to compute accurate estimates and to continue outperforming the benchmark method.

In the forth simulation we added more agents to the network in order to analyze the differences between two different sized networks. So for this simulation the setup was changed. We added two agents to the setup presented in Figure 5.14 obtaining the setup presented in 5.23. For this simulation the target was once again considered to be static.

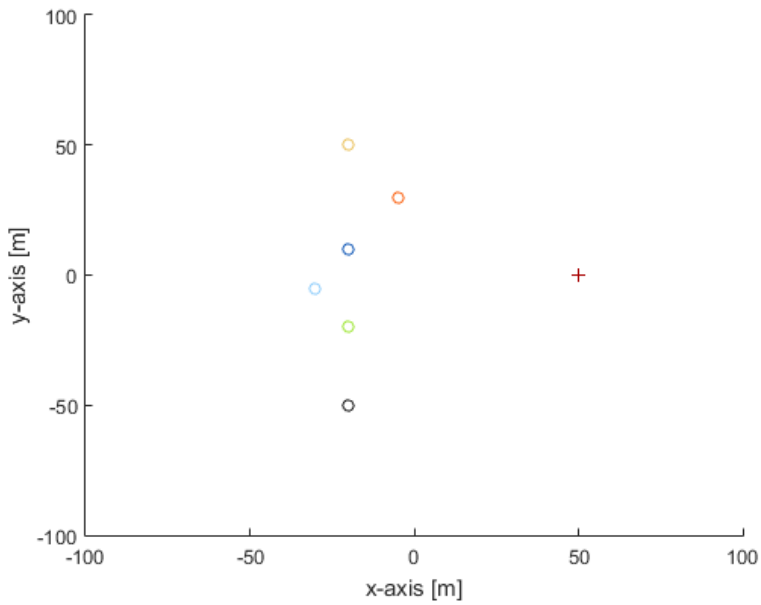
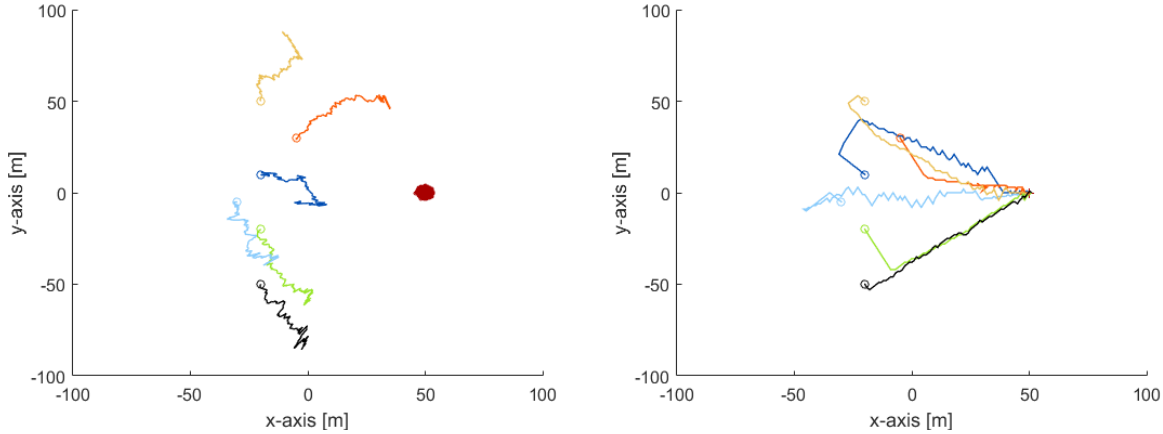


Figure 5.23: Initial setup for the larger network. The circles represent the agents positions and the red cross is the target position.



(a) Benchmark. **(b)** ISEE.U.

Figure 5.24: Obtained trajectories for the larger network simulation for both methods. The trajectories are represented by lines having the same color of the circle of the agent that they correspond. The red line represents the trajectory of the target. The red dots are the samples that represent the belief for the target position in the benchmark method. In **(a)** the agents are spreading at the same time they are going in the direction of the target while in **(b)** the agent move directly to the target.

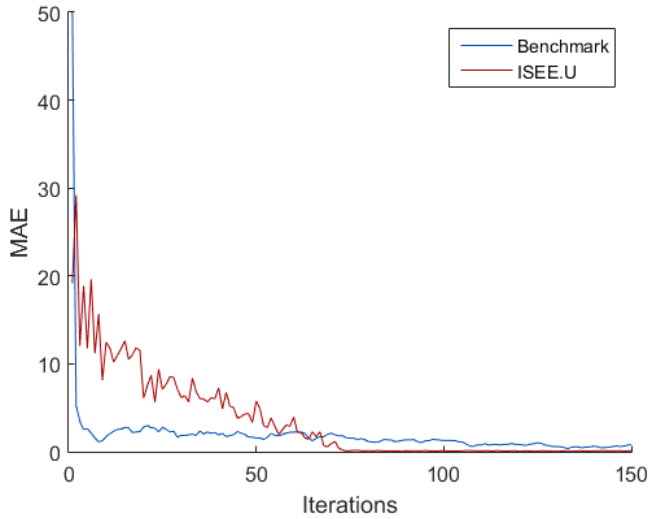


Figure 5.25: MAE obtained for both methods for the larger network simulation. The increase in the number of agents decreases the MAE and ISEE.U outperforms the benchmark after approximately 60 iterations.

In Figures 5.24 it is possible to observe that the obtained trajectories are similar to the ones obtained for the first simulation. The main differences come from the MAE presented in Figure 5.25 since adding more agents decreases the MAE for ISEE.U and it is also possible to observe that the error for ISEE.U reaches the same value as the benchmark in less iterations when compared to the first simulation. This was expected since adding more agents to the network brings also more information about the target position so the estimates are more accurate.

A final test was performed to compare the resources that each algorithm consumed when a simula-

tion was performed. To evaluate that we measured the time that took each algorithm to complete one iteration in the forth simulation (spiral trajectory).

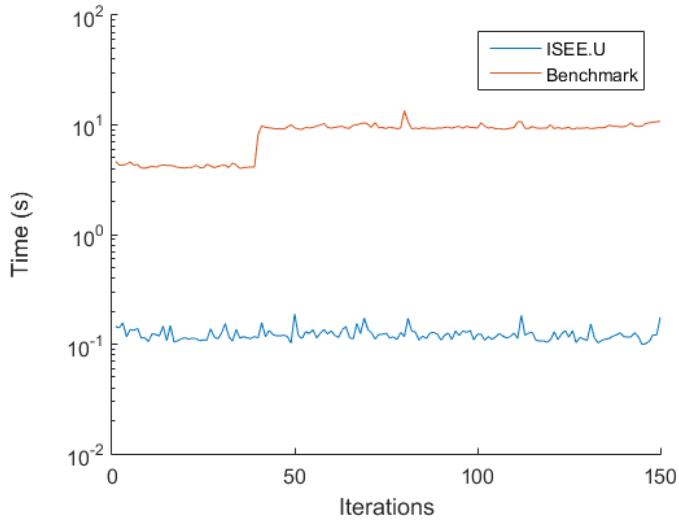


Figure 5.26: Time that each method took to perform one iteration of the simulation. ISEE.U is almost 100 times faster than the benchmark.

From Figure 5.26 it is possible to see that the time ISEE.U takes to make one iteration of the simulation is almost two orders of magnitude smaller than the time the benchmark takes to perform that same iteration. This means that ISEE.U is 100 times faster and so we can conclude that the benchmark uses a lot more resources than ISEE.U.

From all the simulations performed in this Section we can conclude that the method presented in this work showed a very satisfactory performance when compared to a state-of-the-art method supporting the claim that this is a good solution for the problem previously described in Section 1.2.

Chapter 6

Conclusion and Future Work

In this work we proposed two methods to address the problems of estimating multiple target positions using a network of mobile agents in a distributed way, and of computing the control for the movement of the network in order to increase the estimation accuracy. As a byproduct, we developed a consensus + innovations scheme that achieves more precision that the known consensus and consensus + innovations algorithms, for the same number of consensus rounds.

These methods provide approximate solutions for a nonconvex least squares optimization problem by transforming it into a linear problem: one method uses a biased linear estimator to solve it while the other uses the unbiased linear estimator to solve another approximation to the problem.

Our ISEE.U estimator, based on the unbiased linear estimator, is a new consensus + innovations recursion that we proved to be centered. Our ISEE.U estimator, despite not being efficient, could achieve, in our simulations, more precision than the efficient traditional consensus and consensus + innovations for the same number of consensus rounds.

The control takes advantage of two key facts that have gone unnoticed until now: (i) in our distributed scheme every agent has access to an estimate of the covariance matrix for the overall estimator, (ii) the overall covariance can be seen as a sum of each agent's contributions. These two facts imply that an agent can take its estimate of the covariance for the whole network and test which future movement will minimize the uncertainty in the estimation.

When comparing the developed ISEE.U estimator with a state-of-the-art solution under different scenarios, numerical results indicate that, for a static or linear motion target, the benchmark increases localization precision at a faster pace in the beginning of the trajectory, but that ISEE.U catches up and even increases localization precision when compared with the benchmark. For more varied trajectories the results are even more interesting: ISEE.U clearly outperforms the benchmark in localization precision. The running times are also very competitive: ISEE.U takes a few seconds while the benckmark — a solution based on a particle filter — can run for hours.

In conclusion, we were able to find a flexible method that is easy to implement and adapt to very different scenarios and that can still compete in terms of performance with a much more complex state-of-the-art method.

Throughout this work some topics were left unaddressed and they provide great starting points for possible future work.

The first topic is to use the biased linear estimator to solve this problem. ISEE.U and the biased linear estimator were compared and it was possible to conclude that both could provide estimates with similar errors but the bias estimator still outperformed ISEE.U since it considered a better approximation to the original problem. Although this is the best method to solve this problem, it is more complex and a deeper analysis should be made as future work.

The second topic was already mentioned in Section 3.3 and is finding an efficient estimator based on ISEE.U which could perform better than the one presented in this work.

The third topic is to change the cost function that was used to compute the control. As presented in 3.4 we minimize the determinant of the ISEE.U approximation to the covariance matrix which is the same as minimizing the volume of the error ellipsoid. If some eigenvalues of the estimate of the covariance matrix, but not all, are zero the ellipsoid will have zero value making the agents believe that, for that position, the accuracy is maximum, which is an error that was acknowledged in Section 3.4. To solve this error we propose the use of an alternative criterion consisting in the minimization of the maximum eigenvalue of the matrix if one or more eigenvalues are zero which we also acknowledged that is not the best criterion to base our cost function on but solves our problem. Since the probability of an eigenvalue being equal to zero is zero this is not a major problem but still a better solution can be found so that the algorithm is never confronted with a zero eigenvalue. A possible solution is to compute a normalized volume that accounts for eigenvalues equal to zero but still can be compared to other volumes computed from more eigenvalues.

The forth topic is integrating the refinement methods in our solution. As it was presented in Section 5.3 the refinement methods improved significantly the estimate for the target position in the first iterations of the run. Unfortunately, the two methods assume that the target is a cooperative agent that can receive data from the network and adjust the initial estimate given by the ISEE.U estimator to obtain a better estimate. In our setting it is not possible since the target is a non cooperative entity from which we have no information besides the noisy range. To solve this problem we need to find a way to try to substitute the target computations when estimating the target position using, for example, a consensus scheme.

The final topic is adding restrictions to the movement of the network. As one can see in the comparisons between our method and the one chosen as benchmark in Section 5.4 we proposed a way to keep the agents away from the target, still tracking it but without interrupting its movement. This was achieved by creating a safety circle for the target inside which the agents could not enter. As it was seen in the simulations this was a good solution but it still has issues that need to be solved. When the target is moving it can catch an agent inside its safety circle and the agent in its next movement will choose the position around it that minimizes the estimation error but outside of the circle, but since the agent can only move one position per iteration what would happen if all the positions around it are inside the safety circle? In our solution the agent would stay in the same position until one of the possible positions to where it can move is out of the circle, but a better solution can be devised since one agent stays frozen which can, for example, make it disconnected from any other agent. One solution that can

be considered is, given the target position estimates in two consecutive iterations, find the direction of movement of the target and move in a different way in order to get out of the circle. Other solution can be trying to predict the movement of the target and compute the circle for the next predicted positions so the network has enough time to get away from the target but still considering the best positions to guarantee a small estimation error.

Bibliography

[1] F. Gustafsson and F. Gunnarsson, "Mobile positioning using wireless networks: possibilities and fundamental limitations based on available wireless network measurements," *IEEE Signal processing magazine*, vol. 22, no. 4, pp. 41–53, 2005.

[2] A. Beck, P. Stoica, and J. Li, "Exact and approximate solutions of source localization problems," *IEEE Transactions on signal processing*, vol. 56, no. 5, pp. 1770–1778, 2008.

[3] F. Meyer, H. Wymeersch, M. Fröhle, and F. Hlawatsch, "Distributed estimation with information-seeking control in agent networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2439–2456, 2015.

[4] G. Han, J. Jiang, C. Zhang, T. Q. Duong, M. Guizani, and G. K. Karagiannidis, "A survey on mobile anchor node assisted localization in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2220–2243, 2016.

[5] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative robots to observe moving targets," *IEEE transactions on cybernetics*, 2016.

[6] J. Banfi, J. Guzzi, A. Giusti, L. Gambardella, and G. A. Di Caro, "Fair multi-target tracking in cooperative multi-robot systems," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5411–5418.

[7] Y. Ding, M. Zhu, Y. He, and J. Jiang, "P-cmommt algorithm for the cooperative multi-robot observation of multiple moving targets," in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, vol. 2. IEEE, 2006, pp. 9267–9271.

[8] Y. Ding and Y. He, "Flexible formation of the multi-robot system and its application on cmommt problem," in *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, vol. 1. IEEE, 2010, pp. 377–382.

[9] R. Olfati-Saber and P. Jalalkamali, "Coupled distributed estimation and control for mobile sensor networks," *IEEE Transactions on Automatic Control*, vol. 57, no. 10, pp. 2609–2614, 2012.

[10] S. Martínez and F. Bullo, "Optimal sensor placement and motion coordination for target tracking," *Automatica*, vol. 42, no. 4, pp. 661–668, 2006.

[11] H. Wei, W. Lu, P. Zhu, G. Huang, J. Leonard, and S. Ferrari, "Optimized visibility motion planning for target tracking and localization," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 76–82.

[12] H. Wei and S. Ferrari, "A geometric transversals approach to sensor motion planning for tracking maneuvering targets," *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2773–2778, 2015.

[13] G. Soatti, M. Nicoli, S. Savazzi, and U. Spagnolini, "Consensus-based algorithms for distributed network-state estimation and localization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 2, pp. 430–444, 2017.

[14] H. J. Rad, T. Van Waterschoot, and G. Leus, "Cooperative localization using efficient kalman filtering for mobile wireless sensor networks," in *Signal Processing Conference, 2011 19th European*. IEEE, 2011, pp. 1984–1988.

[15] Z. Liu, W. Chen, J. Wang, and H. Wang, "Action selection for active and cooperative global localization based on localizability estimation," in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1012–1018.

[16] I. D. Neveln, L. M. Miller, M. A. MacIver, and T. D. Murphey, "Improving object tracking through distributed exploration of an information map," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 3441–3447.

[17] L. M. Miller, Y. Silverman, M. A. MacIver, and T. D. Murphey, "Ergodic exploration of distributed information," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 36–52, 2016.

[18] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous uav path planning and estimation," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, 2009.

[19] F. Morbidi and G. L. Mariottini, "Active target tracking and cooperative localization for teams of aerial vehicles," *IEEE transactions on control systems technology*, vol. 21, no. 5, pp. 1694–1707, 2013.

[20] S. M. Esmailifar and F. Saghafi, "Moving target localization by cooperation of multiple flying vehicles," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 1, pp. 739–746, 2015.

[21] E. Tzoreff and A. J. Weiss, "Path design for best emitter location using two mobile sensors," *IEEE Transactions on Signal Processing*, vol. 65, no. 19, pp. 5249–5261, 2017.

[22] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability*. Athena Scientific Belmont, MA, 2002, vol. 1.

[23] S. M. Kay, *Fundamentals of statistical signal processing*. Prentice Hall PTR, 1993.

[24] O. Hlinka, F. Hlawatsch, and P. M. Djuric, "Distributed particle filtering in agent networks: A survey, classification, and comparison," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 61–81, 2013.

[25] B. T. Polyak, “Introduction to optimization. translations series in mathematics and engineering,” *Optimization Software*, 1987.

[26] P. Stoica and J. Li, “Lecture notes-source localization from range-difference measurements,” *IEEE Signal Processing Magazine*, vol. 23, no. 6, pp. 63–66, 2006.

[27] A. Winkelbauer, “Moments and absolute moments of the normal distribution,” *arXiv preprint arXiv:1209.4340*, 2012.

[28] Z. Weng and P. M. Djuric, “Efficient estimation of linear parameters from correlated node measurements over networks,” *IEEE Signal Processing Letters*, vol. 21, no. 11, pp. 1408–1412, 2014.

[29] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[30] G. C. Calafiore, L. Carlone, and M. Wei, “A distributed technique for localization of agent formations from relative range measurements,” *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 5, pp. 1065–1076, 2012.

[31] C. Soares, J. Xavier, and J. Gomes, “Distributed, simple and stable network localization,” in *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*. IEEE, 2014, pp. 764–768.

[32] R. Fletcher, “On the barzilai-borwein method,” *Optimization and control with applications*, pp. 235–256, 2005.

[33] J. Barzilai and J. M. Borwein, “Two-point step size gradient methods,” *IMA journal of numerical analysis*, vol. 8, no. 1, pp. 141–148, 1988.

[34] A. Beck and Y. C. Eldar, “Sparsity constrained nonlinear optimization: Optimality conditions and algorithms,” *SIAM Journal on Optimization*, vol. 23, no. 3, pp. 1480–1509, 2013.