Range-Only Node Localization: The Arbitrary Anchor Case In D-Dimensions

Pedro Paulo Ventura Tecchio
University of Pennsylvania

Range-Only Node Localization: The Arbitrary Anchor Case In D-Dimensions

Abstract

This work is situated at the intersection of two large fields of research the Localization problem and applications in Wireless Networks. We are interested in providing good estimations for network node locations in a defined space based on sensor measurements. Many methods have being created for the localization problem, in special we have the classical Triangulation and Trilateration procedures and MultiDimensional Scaling. A more recent method, DILOC, utilizes barycentric coordinates in order to simplify part of the non-linearities inherent to this problem. Except for Triangulation in which we require angle measurements between nodes, the other cited methodologies require, typically only, range measurements. Off course, there exists variants which allow the use of range and angle measurements. We specialize our interest in range only methods utilizing barycentric coordinates by first providing a novel way to compute barycentric coordinates for any possible node-neighbor spatial configuration in any given dimension. Which, we use as basis for our experiments with averaging processes and the development of our centralized and distributed gradient descent algorithms. Our distributed algorithm is able to receive range measurements with noise of uncharacterized distributions as it inputs. Using simulations in Matlab, we provide comparisons of our algorithms with Matlab's MDS function. Lastly, we show our efforts on providing a physical network implementation utilizing existing small form factor computers, wireless communication modules and range sensors.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Electrical & Systems Engineering

First Advisor
George J. Pappas

Keywords
Barycentric coordinates, Cayley-Menger determinants, Localization, Wireless Sensor Networks

Subject Categories
Electrical and Electronics

RANGE-ONLY NODE LOCALIZATION: THE ARBITRARY ANCHOR CASE IN D-DIMENSIONS

Pedro Paulo Ventura Tecchio

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

To my parents

ACKNOWLEDGEMENT

I would like to thank my professors at University of Pennsylvania who guided and taught me all the fundamental knowledge I acquired throughout these last six years. In special, I would like to thank my advisor Professor George J. Pappas, for his help, guidance and patience. He provided me an environment were I could grow as a professional and individual. Moreover, he was a constant among many variables.

I also offer my gratitude to Professor Jean Gallier,for his guidance while I was a student in his class and later on during my Teaching Practicum. And to Professor Victor M. Preciado, from whom I first learned about networks in general and who accepted to be chair of my defense committee.

I thank Nikolay Atanasov and Shahin Shahrampour who I first met as fellow colleagues in the Ph.D. program and with whom I started and continued the work presented in this dissertation. I should also thank them for accepting my invitation to be part of my defense committee as, now, Assistant Professors at University of California San Diego and Texas A&M University respectively.

I also extend my thanks to my therapist Steven Adelman and my psychiatrist Deborah Bernstein. They gave me emotional and medical support during my darkest moments as a normal Ph.D. student.

My deepest acknowledgement and gratitude to my parents Cergio Tecchio and Vera Lucia Ventura who provided me with anything and everything they could during these years.

Lastly, I thank the CAPES / Science Without Borders scholarship program from the Brazilian government that provided me with financial support for four years.

ABSTRACT

RANGE-ONLY NODE LOCALIZATION: THE ARBITRARY ANCHOR CASE IN D-DIMENSIONS

Pedro Paulo Ventura Tecchio

George J. Pappas

This work is situated at the intersection of two large fields of research the Localization problem and applications in Wireless Networks. We are interested in providing good estimations for network node locations in a defined space based on sensor measurements. Many methods have being created for the localization problem, in special we have the classical Triangulation and Trilateration procedures and MultiDimensional Scaling. A more recent method, DILOC, utilizes barycentric coordinates in order to simplify part of the non-linearities inherent to this problem. Except for Triangulation in which we require angle measurements between nodes, the other cited methodologies require, typically only, range measurements. Off course, there exists variants which allow the use of range and angle measurements. We specialize our interest in range only methods utilizing barycentric coordinates by first providing a novel way to compute barycentric coordinates for any possible node-neighbor spatial configuration in any given dimension. Which, we use as basis for our experiments with averaging processes and the development of our centralized and distributed gradient descent algorithms. Our distributed algorithm is able to receive range measurements with noise of uncharacterized distributions as it inputs. Using simulations in Matlab, we provide comparisons of our algorithms with Matlab's MDS function. Lastly, we show our efforts on providing a physical network implementation utilizing existing small form factor computers, wireless communication modules and range sensors.

TABLE OF CONTENTS

LIST OF TABLES

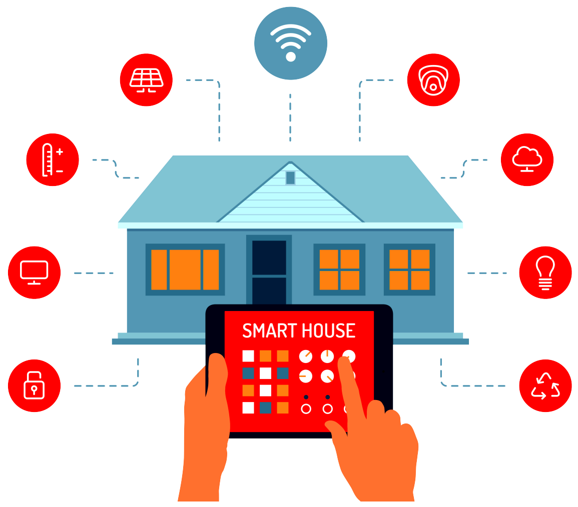
LIST OF ILLUSTRATIONS

CHAPTER 1 : INTRODUCTION

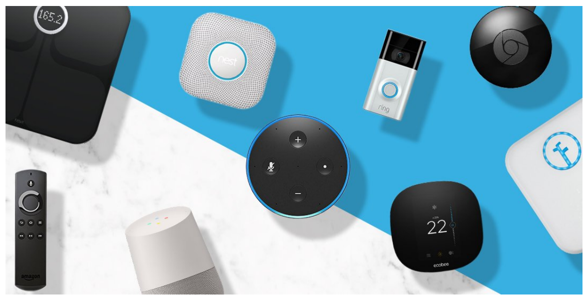
There exists great public interest in large heterogeneous wireless networks. By heterogeneous wireless networks, we mean a wireless network of diverse distinct devices, which may run any kind of software over different hardware platforms. The most common topics involving these networks are IoT - Internet of Things [1,2], VANETs - Vehicular ad-hoc networks [3,4], home automation networks [5–7], mobile robots and robotic sensors [8–10].

One simple example of an IoT is a modern technological driven home, where multiple different appliances and electrical equipment of distinct brands communicate among themselves and/or with user interfaces, usually built as apps in mobile smart-phones, to provide greater comfort and capabilities for homeowners [5–7]. An early implementation of such technology is the popular Nest thermostat witch was first created as a home temperature control center with the ability to learn and to integrate via WiFi to other devices. Later, other products from the same company were marketed such as smoke and CO_2 detector and a complement of interconnected cameras. Nowadays, the brand and technology is owned by the Google group under their holding company Alphabet Inc and these devices are marketed to work in an environment of up to 3000 different products including Google's own voice interface ”Hello Google” which enables them to use voice control. Another leader in this segment is Amazon.com Inc. which similarly works in partnership with other companies to enable its own ”Alexa” voice interface to control different devices around the same WiFi or Bluetooth networks.

Also in the segment of home automation, we can attach personal health care devices and products for personal everyday use such as electric toothbrushes and Bipap machines which interconnect with smartphones in order to provide historical data and a user friendly control interface. In a more extreme case, we can include future deployment of mobile autonomous robots for the health care of its residents [11–13]. We believe that such robots will be extensively used in everyday homes were the care of seniors and infants are re-



(a) Smart-home conceptions [14].

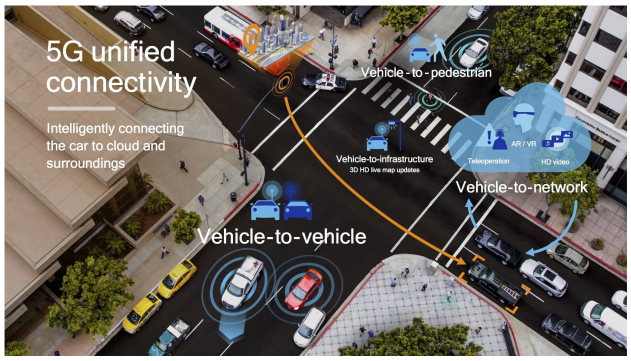


(b) Smart-home existing devices [15].

Figure 1: Fig. 1a shows the general areas in which home automation is sought using wireless node networks such as security, devices, temperature, illumination and cloud services control. Meanwhile, Fig. 1b shows devices already in large scale production which seek to provide home automation features at least in determined areas, for example we mention Google Nest and Amazon.com Echo.

quired, specially for tasks which are considered a chore by their families or in which the family members are not trained to perform or are not able to, as examples we cite cleaning of human waste, application of intravenous medicines or physically lifting impaired people.

Meanwhile, vehicle-to-vehicle and vehicle-to-roadside communication architectures are the fundamental blocks of VANETs. These communication protocols are being built in order to enable new vehicle features and road services, such as vehicle security through shared information about threats, traffic management and better performance and reliability of self-driven cars [3, 4, 16–20]. Up to the publication of this dissertation, there is no common communication and sensing network protocols and standards, as such technologies are being extensively researched and implemented by numerous vehicles manufactures, technological companies and academy. The autonomous car may not be the hot topic of the moment in the academic world, but it is still in the process of being studied and readied for mass production and everyday utilization. It is a common goal that autonomous cars



(a) 5G enabled communication infrastructure [21].

(b) Self driven car [22].

Figure 2: The future communication infrastructure based on 5G devices will allow for heterogeneous devices to intercommunicate, Fig. 2a, while a conceptual example of a self driven car is shown in Fig. 2b.

will be able to self coordinate and also plan their operation in a cooperative way with other cars and with the road infrastructure enabling for example local and global traffic management and control; road alerts propagated throughout the network so that cars can plan their actions in advance, and vehicle platooning in small (road-wide) to large (city-wide) formations. Moreover, in the near future, maybe a few decades or so, vehicle accidents with injured or dead passengers or pedestrians will be seldom seen, as there will exist less and less human interference in the system with increasing redundancy of sensors, network communication and system intelligence.

Even if one may not consider future vehicles as mobile robots and we may also disregard home automation robots, one may not forget about the large insurgency of drones that is happening now. Major and minor companies are studying ways to apply drone technology to either its operations or products. A famous example is again Amazon.com Inc. which seeks to utilize drones as a mean of package delivery in highly populated cities. Another major segment in which drones are being researched for and applied are law enforcement, rescue and military. In these, drones may be used for exploration, infiltration, target search and track, target pacification, packages, objects and ordnance delivery among others. Meanwhile, their utilization has been a topic of concern among some legislators and the public in general as they can infringe on civil rights when coupled with cameras and

(a) Mobile robots used in undersea exploration [23].

(b) Mobile robost used for site security [24].

Figure 3: Examples of wireless networked mobile robots used for exploration Fig. 3a and for security Fig. 3b.

other recording devices; cause accidents and property damage and also be used for nefarious and unlawful activities.

While many devices and applications are already being marketed and used in somewhat niche markets, most of them are still being researched and developed. Among the different lines of research and development, our interest lies in the exploration of a topic that they usually have in common, the need to known where they are being used. This need may be a focus of interest among device users and maintainers or among the devices themselves.

The ability of localize oneself in the environment is an important feature to many systems, as it may influence its behavior and functionality. A human placed in an unknown environment will have difficulties to live in it effectively, the same can be said to a sensor or robot, as they will possibly not be able to effectively perform their tasks. Besides the earlier examples, we can mention a simple one: a network of atmospheric sensors used to obtain climate data for general use. If one does not known the position of these sensors, their gathered information becomes almost meaningless. It is a major problem to know a hurricane is coming but not knowing where it is and to where it is going. One may mention that in such an example, all sensors will be placed in fixed and GPS enabled locations, so one can measure once and use the data forever. But what about networks in which such a case does not happen, for example inside a building or with moving nodes?

As can be inferred, our interest lies in localizing nodes of a network, more specifically static ones which are able to range themselves in relation to their neighbors. We focus on static networks as they provide an entry point for the localization problem and are also largely used in homes, business and industrial applications. On another hand, we were able to supplement the existing theory of static network localization in a meaningful way, by providing a formulation that works in any possible spatial dimension and with fewer constraints on the network. Regarding the utilization of range measurements alone, one point of view it that having both range and bearing measurements among nodes may provide a larger amount of information, another is that the restriction to range only measurements should be seen as a feature as we only require partial information about inter node positioning and therefore less sensors. Having said so, we do not mean to imply we are not interested in studying the possibility of integrating bearing measurements to our methodology in the near future, as it is possible that having both will enable us to create a more efficient method.

For now, our proposed method is able to localize nodes in a static network provided that we give the actual and noiseless location of $d+1$ nodes of the network in d-dimensional Euclidean space (which are called anchor nodes), the range measurements between all neighboring nodes and that each node must have at least $d+1$ neighbors. It may be the case, that some nodes satisfy these constraints, but are not localizable, which will happen if their neighbors are not localizable themselves. So an unlocalizable node may affect an entire segment of a network. Therefore, localizable nodes form a network with anchor nodes, in which, nodes have at least $d+1$ neighbors. In spite of those requirements, there is no restriction on the position of nodes in relation to anchors as exists among other methods which we will specify in the next section. This is an important feature, as it enables the utilization of our algorithm to localize more diverse networks regardless of the chosen location for anchors, an easy example is given in 3-dimensional space (3-D), where other methods would require two sets of 4 anchors, in which case at most 3 may be used in both sets, and two runs of the algorithm to localize nodes in a rectangular prism shaped environment (a common room rectangular floored room), ours would only require the application of the algorithm once with 4 anchors.

We formalize our localization problem in Chapter 2, then we explain our theoretical reasoning in Chapters 3 and 4. In these chapters, we develop our method based on Barycentric Coordinates, which may be easily understood as computing proportional volumes of region segments in space defined by the location of their vertices. In other words, given a d-dimensional space, one region in it is defined using $d+1$ points as vertices of this region, which actually is a simplex. Then by choosing a different point in the same space and computing the volume of each possible simplices formed by substituting one point of the previous set with this new point, we can compare the volumes of the new regions with the old one by division, which is exactly how barycentric coordinates are calculated. Therefore, in the case of a network, each node is treated as a point in space with a Cartesian coordinate, then we choose $d + 1$ neighbors of this node which form a simplex and compute the barycentric coordinates for each node in relation to each chosen subset of its neighbors. These coordinates form a linear system of equations where the node Cartesian coordinates are the unknown variables. Therefore, by isolating the anchor coordinates which are known from the rest of unknown variables, one may solve this linear system and compute all unknown node coordinates.

One may ask, how do we compute the simplices volumes if we are only measuring inter-node ranges? If one consult the existing literature, one will probably find the concept of Cayley-Menger determinant. This determinant relates the Eucldiean Distance Matrix, matrix formed by squared Euclidean distances between each pair of points in a given set of $d + 1$ elements in d-dimensional Euclidean space, with the simplex volume formed by such points squared. As the Euclidean distance is the same as the range between the nodes, one has a functional relation between range measurements and barycentric coordinates. But there is a catch is this relation. As mentioned, the value obtained from computing Cayley-Menger determinant is proportional to the square value of the simplex volume, so one does not know which sign, positive or negative, to use in order to compute barycentric coordinates.

If one restrict its nodes to always be inside the convex hull of their neighbors, and by extension, the entire network should be inside the convex hull of the anchors, one is able to ignore this sign problem, as the correct sign should always be positive in this case by construction. This fact is explored by the vast majority of network localization works that exist nowadays. Other works are either restricted in the space dimension or use mixed approaches to solve the localization problem. We on the other hand, developed a method to compute barycentric coordinates with their correct signs by applying a more general form of the previous determinant called Cayley-Menger bi-determinant, which is not so commonly found in the related literature. This bi-determinant takes to distinct sets of points and compute a value proportional to the multiplication of their simplices volumes, which we can use to compute barycentric coordinates and proceed as previously explained in other to compute the unknown node coordinates.

So far, we talked about computing the Cartesian coordinates without mentioning anything about noise or how does our algorithm operates. It is because this theory enables us to solve the problem in centralized form with noiseless range measurements, but what happens when that is not the case? Adding noise to our measurements becomes a mess, even using simple zero mean additive Gaussian noise to each measurement makes the output of the Cayley-Menger bi-determinant have an unknown probability distribution with complex terms formed by sums of products and divisions of Chi-Squared random variables. So the standard approach to solve this problem in closed form becomes unfeasible. Following, some related works we decided to apply an averaging process in all range measurements individually, as we consider that they will have at most zero mean random additive noise applied to them. Note that we are not assuming any specific kind of distribution here. Then, by the Law of Large Numbers the average should converge to an approximate noiseless range value when one approaches a sufficiently high number of noisy measurements. Therefore, in theory in order to easily solve this problem, one should acquire a large number of measurements, compute the Cayley-Menger determinants, compute barycentric coordinates and solve the linear system of equations arriving to a probably good enough location estimates for each unknown node. As always, in practice it does not always work. Numerical errors from all non linearly computation may lead to high discrepancies in the actual computed values for the barycentric coordinates from one set of measurements to another. The easiest solution was to average the barycentric coordinates again in order to suppress such problems. As we did so, we arrive to our first centralized method able to provide somewhat good estimates over time.

In order to arrive to a distributed solution, we first experimented with a centralized gradient descent algorithm that worked as intended with both fixed and dynamic step sizes. Then, we modified our mathematical formulation to apply an existing theoretical algorithm to solve optimization problems in distributed systems over a given time horizon. This choice proved fruitful as we were able to finally solve our localization problem with noisy range measurements in a distributed form. Simulated experiments where we applied our algorithms to randomly generated networks will be shown in Chapter 5 with the respective discussions.

Lastly, during our theory development, we also sought to provide a physical platform in order to demonstrate our work, but we were not completely successful in this front. As will be discussed in Chapter 6, we provide in Appendix A.2.1 a recipe of sorts on how to compile, configure and apply a Linux distribution with the correct drivers in order to use our chosen platform and sensor suite with the necessary network configuration and data extraction software. Our platform was constructed around a Beagle Bone Green Wireless board with integrated WiFi module capable of operating in mesh networks. We notice that this was the first board in the market able to do so, while nowadays there may be other possibilities to choose from. At first, we used a range sensor called Polypoint, but it operated in star topology which disallowed us to use it in real time operations as one had to keep switching it on and off to change its configuration from anchor to unknown node, *i.e.* in the entire sensor network only one device ranged itself at a given time in relation to all others, so it was required to reset it every time one wanted obtain range measurements from other devices. Because of this constraint, we sought other sensors without much success. At one point, we finally decided to use RSS - Received Signal Strength measurements from the WiFi module as a means of estimating range measurements. We did some experiments at first in open space without occlusions and the results were satisfactory, but later on when working inside of a building with walls and other objects cluttering the space in which the sensors were positioned, our first measurement model became irrelevant. Even a change in the sensor height in relation to the ground would change the RSS measurements received. Therefore, lacking an appropriate range sensor, we decided to focus on the theoretical aspect of the problem.

In the next section we will introduce the relevant existing works and start introducing the range only localization problem with arbitrary anchor placement in a more formal setting.

1.1. Related work

An heterogeneous wireless network must satisfy a series of constraints in order to work correctly as intended by its stakeholders, be its developers, maintainers and users. One main aspect of such networks is that it may contain both static and or mobile nodes which are loaded with sensors necessary for their independent or communal operations. In either case, it is most likely that their location will influence their operation and *vice versa*. Therefore, it is important for each node in a network to know its current location over time with respect to some local or global frame of reference. Relative localization is especially challenging but also most important to solve robustly in harsh operational conditions, where GPS access is denied, the surroundings are featureless, and the sensed information is lower-dimensional than the sensor states of interest (*e.g.* bearing-only or range-only measurements used for 3-dimensional (3-D) position and orientation estimation) and perturbed by significant measurement noise. To correctly determine a frame of reference, it is usually required that locations of a small subset of its nodes, called anchor nodes, are known. Throughout this paper we call all other nodes unknowns as they are not localized.

The localization problem of wireless networks has been considered in 2 and 3 dimensions, with some working in higher generic dimensions, using different measurement types and noise distributions. Range and bearing measurements are utilized in some methodologies [25–27], while others rely only on one of them such as bearing-only [28–31] and range-only measurements [32–38]. Typically most algorithms start in a centralized fashion [39–42], *i.e.* all network information is sent to a ”central” node responsible for more intense computational needs to localize all network nodes; else, if each node remains responsible for its own localization based on shared local information, the algorithm is called distributed [27, 30, 32–35, 38, 43–58].

Multiple different methodologies have been developed over years of research in localization. One of the most fundamental methods is based on Triangulation [40–42] or more commonly Trilateration [8, 59]. The later can be summarized as the iterative resolution of a non-linear system of equations relating the range measurement between nodes and the Euclidean distance between the location estimates for each unknown node. The 2-D example below demonstrate the set of equations needed to localize node l in relation to nodes i, j, k:

$$\begin{cases} ||\mathbf{x}_l - \mathbf{x}_i||_2 &= z_{li} \\ ||\mathbf{x}_l - \mathbf{x}_j||_2 &= z_{lj} \\ ||\mathbf{x}_l - \mathbf{x}_k||_2 &= z_{lk} \end{cases} \tag{1.1}$$

As mentioned, Trilateration methods work by localizing one unknown node at each iteration based on existing anchor nodes or already localized unknown nodes, which is an iterative process. Different from this, there are methodologies in which all necessary information is gathered and the localization of all nodes is done simultaneously either in a centralized or distributed form. We cite Kalman based algorithms [27,31,60], Multilateration [61,62], MDS

- MultiDimensional Scaling [35, 39, 63, 64] and barycentric coordinates based methods [32–35, 48, 50–52, 54]. From these, we emphasize both MultiDimensional Scaling and barycentric methods. We utilize MDS as a benchmark tool for our proposed algorithm which is based on barycentric coordinates. The classical MDS approach is to solve an optimization problem minimizing the following function:

$$\textit{Stress}(\mathbf{x}_1, \cdots, \mathbf{x}_n) = \left(\frac{\sum_{i,j} (Z_{ij} - ||\mathbf{x}_i - \mathbf{x}_j||_2)^2}{\sum_{i,j} Z_{ij}^2} \right)^{1/2} \tag{1.2}$$

In spite of not having a convex nor linear function to be optimized, MDS methods have an advantage: they can be solved through the manipulation of its algebraic equations in matrix form, so that solving the minimization problem becomes equivalent to eigenvalue and eigenvector decomposition of a matrix, for which exists optimized algorithms. Their disadvantage is that their solution is dependent of initial estimates. Therefore, using random initialization may incur on solutions with stress varying from small to large values, so one needs to run the algorithm multiple times in order to get a good solution.

The barycentric coordinates approach seeks to hide all non-linearities of this problem in the computation of barycentric coordinates. So, after one finds all barycentric coordinates for all nodes in the network, the solution of the noiseless system becomes as easy as solving a linear system. This method was first proposed by Khan *et al.* in [32]. It is interesting to notice that the system becomes automatically distributed along the many nodes of the network, *i.e.* each node is able to compute its barycentric coordinates with local information.

This method becomes harder to solve whenever there is noise added to the range measurements, as one need to pay attention to the errors inserted on the computation of the heavily non-linear set of equations involved on the computation of all barycentric coordinates. Moreover, because of the same non-linearities, there is no closed form solution for barycentric coordinates given even simple additive Gaussian noise to range measurements. In order to solve this last problem, Khan *et al.* proposed an algorithm based on averaging processes [34].

Multiple variations of the first barycentric algorithm [32] were proposed such as [33–35, 48–52, 54]. Khan *et al.* in special provide us with a diverse collection of works extending this framework in static and mobile networks and in different applications where unknown nodes are strictly inside the convex-hull of anchors [32–34, 48, 49, 52]. In another direction, Diao *et al.* proposed in [33] an algorithm for the 2-D case, that allow an arbitrary positioning of anchors among unknown nodes. Unfortunately, their line of reasoning to solve the localization problem was based on the geometrical identification of all conditions necessary to correctly compute barycentric coordinate's signs in more than 2-D. In [35], Han *et al.* proposed a hybrid approach of mixing barycentric coordinates with MDS to solve the same problem in 3-dimensions.

In this work, we leverage the definition of Cayley-Menger bi-determinants found in [8] to deduce a new formulation of barycentric coordinates which allow us to extend the previous results of Diao *et al.* to the generic d-dimensional space. As will be shown, we provide a distributed algorithm able to localize nodes in d-dimensional Euclidean space with only inter-node noisy measurements and a small subset of $d + 1$ anchors. In our algorithm, there is no assumption about the position of any node in the network; *i. e.* anchors can be anywhere in relation to unknowns. Moreover, we do not assume any previous knowledge about noise characteristics of any range measurement. Locations of anchors which should be noiseless are an exception, though.

Excluding [35], which utilizes a hybrid approach of using barycentric coordinates and MDS - MultiDimensional Scaling; previous works known to use barycentric coordinates, either restrict the dimension of the problem to two [33] or assume some specific arrangement for anchor and unknown nodes [32, 34, 48, 49, 52]. While our approach may still present some inconveniences, such as longer run time than MDS, in a stricter point of view, we provide a pure way to solve the localization problem of any d-dimensional static node network using barycentric coordinates with range measurements and $d+1$ node locations.

Even though, by our knowledge, no practical application for localization problems in greater than 3-dimensional Euclidean spaces exists nowadays; we still believe it is necessary to provide the related theory for such a case, as it may occur in the future.

Succinctly, our main contributions are:

- Provide a way to compute barycentric coordinates for any node arrangement in d-dimensional Euclidean space.
- Provide a centralized algorithm able to solve the localization problem of d-dimensional static networks based on noiseless range measurements and $d+1$ node locations.
- Provide a distributed algorithm able to solve the previous localization algorithm under noisy range measurements.

CHAPTER 2 : PROBLEM DEFINITION

A static sensor network in d-dimension Euclidean space can be modeled as a weighted directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{Z}\}$, with vertex set $\mathcal{V} := \{1, 2, \ldots, n\}$. From this set, we define the subset $\mathcal{A}$ of anchor nodes with known locations and the subset $\mathcal{U}$ of nodes with unknown locations, such that $\mathcal{V} = \mathcal{A} \bigcup \mathcal{U}$ and $\mathcal{A} \bigcap \mathcal{U} = \emptyset$. We refer to the Cartesian coordinates $\mathbf{x_i} \in \mathbb{R}^d$ of node $i \in \mathcal{V}$ as its location. Therefore, the set of all node locations is $\mathcal{X}$, of anchor nodes is $\mathcal{X}_{\mathcal{A}}$ and of unknowns is $\mathcal{X}_{\mathcal{U}}$.

An edge $(i, j) \in \mathcal{E}$, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, exists whenever nodes i and j are able to communicate and measure their relative distances, $d(\mathbf{x_i}, \mathbf{x_j}) = ||\mathbf{x_i} - \mathbf{x_j}||_2$. Supposing that each node $i \in \mathcal{V}$ has a maximum sensing range $r_i \in \mathbb{R}_{>0}$, an edge between node pair (i, j) exists if $d(\mathbf{x_i}, \mathbf{x_j}) \leq min\{r_i, r_j\}$.

An example of such a network in 3-D is shown in Fig. 4. In this example, all 64 nodes are arranged in a cubic lattice with a lattice constant of 1 unit. Inter-node measuring and communication ranges are 2 units, $r_i = 2$, $1 \leq i \leq m$. Fig. 4 also shows the node degree[1] of each vertex in this network example.

We suppose that each pair of nodes $(i, j) \in \mathcal{E}$ obtains range measurements over time $t = 1, \ldots, T$, which can be arranged in a set of square matrices such as $\mathcal{Z} := \{Z_1, Z_2, \ldots, Z_T\}$ with $Z_t \in \mathbb{R}^{n \times n}$ $\forall t$. Moreover, $[Z_t]_{ij} = d(\mathbf{x_i}, \mathbf{x_j}) + \delta_{ij}(t)$ where $\delta_{ij}(t)$, for all pairs $(i, j) \in \mathcal{E}$ in time, are independent identically distributed (iid) random variables with unknown probability distribution. Assuming $\delta_{ij}(t)$ and $\delta_{ji}(t)$ are independent, the measured quantities $[Z_t]_{ij}$ and $[Z_t]_{ji}$ are distinct random variables and thus the generated graph is directed. Conversely, the noiseless case leads to an undirected graph formulation.

Therefore, based on the previous concepts, we can formulate the general form of the localization problem as the following:

[1] In graph theory, node degree refers to the number of edges a node has to other nodes in the network, which are its direct neighbors.

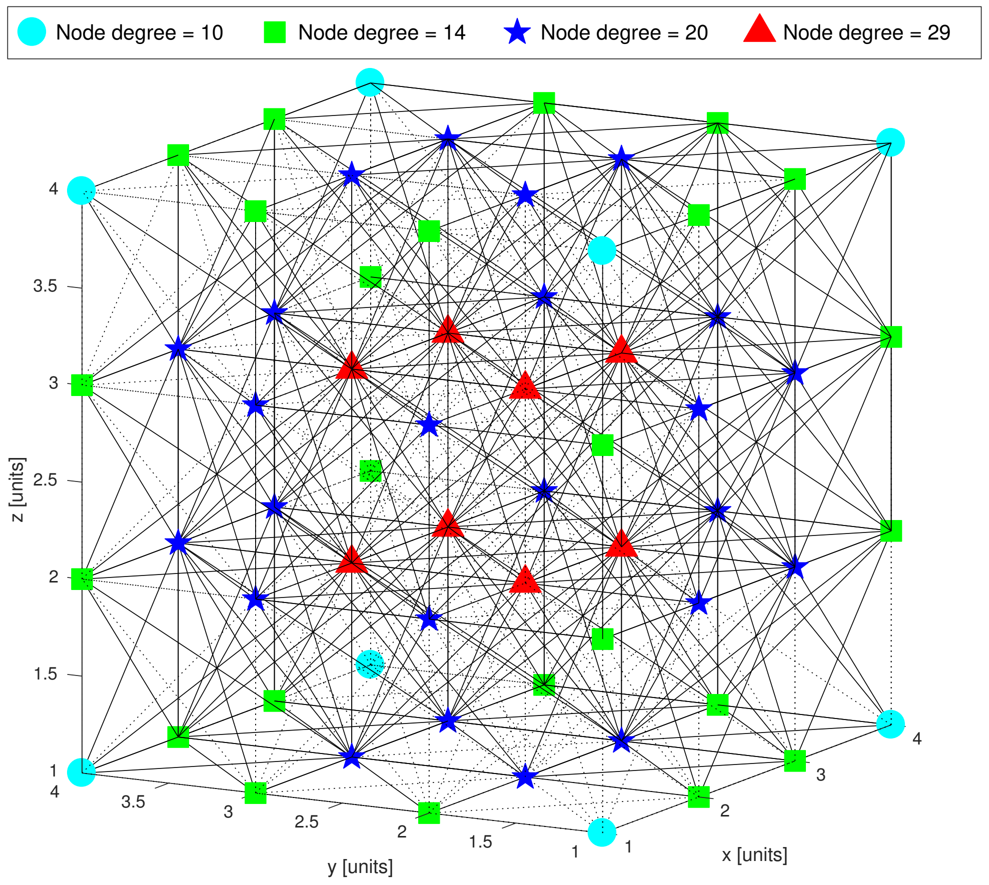


Figure 4: Wireless node network example. Nodes are arranged in a cubic lattice with lattice constant of 1 unit. We chose an inter-node measuring and communication range of 2 units for all nodes. The different vertex colors indicate nodes with different node degrees as specified in its legends.

Problem. *Given the anchor node locations $\mathcal{X}_{\mathcal{A}} \subset \mathcal{X}$ of a static sensor network and noisy range measurements $\mathcal{Z}$, estimate the locations of all unknown nodes $\mathcal{X}_{\mathcal{U}} \subset \mathcal{X}$.*

In the next section, we define Cayley-Menger bi-determinants and barycentric coordinates. Then, we show how to use the first to compute the second for any possible node arrangement. Next, based on barycentric coordinates, we show how to solve the centralized noiseless case of the generic localization problem and later the distributed noisy case. But first, let us summarize in Table 1 our most used symbols with their meaning for future reference.

,n

Table 1: Symbol notation

Symbol	Meaning
i, j, k	Generic indices
l	Index referencing the l^{th} node
t	Index referencing time iteration
d	Number of dimensions
n	Number of nodes $n := \|\mathcal{V}\|$
a	Number of anchor nodes $a := \|\mathcal{A}\|$
u	Number of unknown nodes $u := \|\mathcal{U}\|$
r_l	Maximum sense range of node l
$\delta_{ij}(t)$	Random noise at edge (i, j) at time t
$\mathbb{K}_n$	Group of complete graphs with n vertices
$\mathcal{P}$, $\mathcal{Q}$	Generic sets
$\mathcal{V}$	Set of vertices or nodes
$\mathcal{E}$	Set of edges
$\mathcal{Z}$	Set of range measurement matrices
$\mathcal{A}$	Subset of anchor nodes
$\mathcal{U}$	Subset of unknown nodes
$\mathcal{N}^{(l)}$	Set of neighbors of node l
$\mathcal{I}^{(l)}$	Family of sets of $d+1$ indexes given by all possible combinations of $d+1$ neighbors of node l which form a subgraph in $\mathbb{K}_{d+1}$
$\mathcal{X}^{(l)}$	Family of sets of Cartesian coordinates of neighbors of l given by $\mathcal{I}^{(l)}$
$\mathcal{X}_i^{(l)}$	i^{th} set of family $\mathcal{X}^{(l)}$
$\widehat{\mathcal{X}}_{i_j}^{(l)}$	i^{th} set of family $\mathcal{X}^{(l)}$ with j^{th} element modified
$\mathbf{p}, \mathbf{q}$	Generic real column vectors with some specified dimension
$\mathbb{1}$	Vector of ones with correct dimension
$\mathbf{x_l}$, $\mathbf{x}^{(l)}$	Location of node l in $\mathbb{R}^d$
$[\mathbf{x_l}]_i$, $\mathbf{x}_i^{(l)}$	Element at i^{th}-row of vector $\mathbf{x}^{(l)}$
P, Q	Generic matrices with some specified dimension
X	Location matrix in $\mathbb{R}^{d \times n}$
Z_t	Range measurement matrix at time t
L_l, $L^{(l)}$	Matrix of barycentric coordinates of node l
$[L_l]_{ij}$, $L_{ij}^{(l)}$	Element at i^{th}-row and j^{th}-column of matrix $L^{(l)}$
$\mathbb{L}^{(l)}$	Generalized barycentric matrix (GBC) of node l
$\mathbb{L}_{:\mathbb{A}}^{(l)}$	The columns of GBC of l that are indexed by $\mathcal{A}$
$\mathbb{L}_{:\mathcal{U}}^{(l)}$	The columns of GBC of l that are indexed by $\mathbb{U}$

CHAPTER 3 : A NOISELESS CENTRALIZED SOLUTION

We start our endeavor to solve the localization problem proposed in Chapter 2 in a centralized form under noiseless range measurements by defining the theoretical background on Cayley-Menger determinants, bi-determinants and barycentric coordinates. These concepts will prove to be fundamental to our proposed centralized algorithm, as one can utilize Cayley-Menger determinants and bi-determinants to compute barycentric coordinates and with them one is able to solve the localization problem by simply computing the solution of a linear system.

Cayley-Menger determinants were extensively used by Blumenthal in his book about Distance Geometry [65], were he defined and proved this determinants property of relating Euclidean Distance Matrices proportionally to the square of the signed volume of the simplex formed by the set of $d+1$ points utilized in the Euclidean Distance Matrix in d-dimensions. Then, we summarize how these concepts were applied under different assumptions by Khan *et al.* in [32] and by Diao *et al.* in [33], which we consider to be the precursors of our solution. The former is the first work to utilize barycentric coordinates to solve range-only localization in wireless networks, while the later is an extension or modification of the former seeking to allow arbitrary placement of anchors among unknown nodes in the two dimensional case. By arbitrary placement of anchor nodes, we mean that anchor and unknown nodes may be placed randomly, but they must satisfy the criterion of having at least $d+1$ distinct neighbors that form a simplex of non-zero volume. This is an extension on Khan's work which required all nodes to be place inside the convex-hull of a subset of their neighbors and consequently, to have all unknown nodes inside the convex-hull of the $d+1$ anchor nodes.

In [8], we discovered the concept of Cayley-Menger bi-determinants without any formal proof, which we later constructed based on Blumenthal's proof of the determinant, and explored its utilization to compute barycentric coordinates instead of using the simpler determinant. This choice proved extremely successful, as it allowed us to develop a computationally efficient way to calculate correctly signed barycentric coordinates for any set of $d+1$ point in d-dimensional Euclidean space based on their Cartesian coordinates. The ability to compute all signs is fundamental to our work as it enables us to provide an alternative solution for Diao's work in the 2-D case, moreover it also allowed us to extend their idea to any possible dimension. It is not as fundamental to works which follow Khan's assumption of having nodes inside the convex-hull of their neighbors, as in this case all signs are strictly positive.

Later on we provide the best possible least squares estimate for our proposed localization problem via the solution of a over-determined solution, moreover if the inverse of a sub-matrix of this system exists, then we are able to compute the correct and full solution for each unknown node Cartesian coordinate.

3.1. Fundamental theory

3.1.1. Cayley-Menger determinant

Cayley-Menger determinants, [32–34, 65], provide a relation between Euclidean distances of points in space and the signed volume of the simplex formed by such points as seen below in Definition 1 and Theorem 1.

Definition 1. *Let the tuple of $d+1$ points, $\mathcal{P} = (\mathbf{p_0}, \ldots, \mathbf{p_d})$, be defined by its Cartesian coordinates, such that $\mathbf{p_i} \in \mathbb{R}^d$ for $0 \leq i \leq d$. Then, the Cayley-Menger determinant of $\mathcal{P}$ is defined as follows:*

$$D(\mathcal{P}) = D(\mathbf{p_0}, \ldots, \mathbf{p_d}) = \\ 2\left(-\frac{1}{2}\right)^{d+1} \begin{vmatrix} 0 & 1 & 1 & \ldots & 1 \\ 1 & 0 & d(\mathbf{p_0}, \mathbf{p_1})^2 & \ldots & d(\mathbf{p_0}, \mathbf{p_d})^2 \\ 1 & d(\mathbf{p_1}, \mathbf{p_0})^2 & 0 & \ldots & d(\mathbf{p_1}, \mathbf{p_d})^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d(\mathbf{p_d}, \mathbf{p_0})^2 & d(\mathbf{p_d}, \mathbf{p_1})^2 & \ldots & 0 \end{vmatrix}. \tag{3.1}$$

Theorem 1. *The Cayley-Menger determinant of a tuple of $d+1$ points in $\mathbb{R}^d$, $\mathcal{P} = (\mathbf{p_0}, \dots, \mathbf{p_d})$, is related to the square of its signed volume as in*

$$D(\mathcal{P}) = (d!)^2 \textit{Vol}(\mathcal{P})^2. \tag{3.2}$$

Proof. See Appendix A.1 by taking $\mathcal{Q} = \mathcal{P}$. □

But, why is this Cayley-Menger determinant so important for our problem solution? One may reply, that by actually changing the computed Euclidean distances of points in $\mathcal{P}$ to actual range measurements $[Z_t]_{ij}\, \forall (i,j) \in \mathcal{E}$, one may actually compute the experimental signed volume squared, which will be used to compute barycentric coordinates in the next section.

3.1.2. Applying barycentric coordinates

Barycentric coordinates are usually defined using concepts of points, affine spaces and affine frames. An affine space $\mathbf{P}$ is defined by a collection of points, a vector space and a function. An affine frame is a set of points in an affine space with origin $\mathbf{p_0}$, $\{\mathbf{p_0}, \mathbf{p_1}, \cdots, \mathbf{p_d}\}$, such that vectors $\{\overrightarrow{\mathbf{p_0 p_1}}, \overrightarrow{\mathbf{p_0 p_2}}, \cdots, \overrightarrow{\mathbf{p_0 p_d}}\}$ are linearly independent, i.e. they form a base for the embedded vector space $\overrightarrow{\mathbf{P}}$. By taking a field $\mathbf{K}$ such as $\mathbb{R}$, one can define barycentric coordinates as follows:

Definition 2 ([66, Prop.3.6.2 Modified])**.** *Let $\{\mathbf{p_i}\}_{i=0,1,\dots,d}$ be a frame for an affine space $\mathbf{P}$. For any (point) $\mathbf{p} \in \mathbf{P}$ there exist $\lambda_i \in \mathbf{K}$, $0 \leq i \leq d$, such that $\sum_i \lambda_i = 1$ and $\mathbf{p} = \sum_i \lambda_i \mathbf{p_i}$. The scalars λ_i are uniquely defined by this property and are called barycentric coordinates of $\mathbf{p}$ in the frame $\{\mathbf{p_i}\}_{i=0,1,\dots,d}$.*

From Definition 2, one can infer that in order to localize an unknown node in the network, one needs to know the locations of $d+1$ nodes in an d-dimensional space which form an affine frame. Therefore, for each network one needs at least $d+1$ anchor nodes which form an affine frame for the inherent affine space. Also, from Definition 2 and Theorem 1 one may compute the **absolute** value of any barycentric coordinate from a tuple of points in space as given next:

Definition 3. *Let the tuple $\mathcal{P} = (\mathbf{p_0}, \cdots, \mathbf{p_d})$ be a frame for an affine space $\mathbf{P}$. Then, given a point $\mathbf{p} \in \mathbf{P}$, the **absolute** value of its barycentric coordinates in relation to the tuple $\mathcal{P}$ is:*

$$|\boldsymbol{\lambda}_i| = \sqrt{\frac{D(\mathbf{p_0}, \ldots, \mathbf{p_{i-1}}, \mathbf{p}, \mathbf{p_{i+1}}, \ldots, \mathbf{p_d})}{D(\mathbf{p_0}, \ldots, \mathbf{p_d})}} \tag{3.3}$$

Later, in Section 3.3.2 we will provide a formal and novel way to compute the true signed barycentric coordinates for any point $\mathbf{p} \in \mathbf{P}$ in relation to the tuple $\mathcal{P}$. So far, in order to present the aforementioned summary of Khan *et al.*'s DILOC, [32], and Diao *et al.*'s ECHO, [33], the existing definitions will suffice.

3.2. Existing methods

3.2.1. DILOC

In [32], Khan *et al.* proposed the DILOC algorithm utilizing barycentric coordinates to solve localization problems in wireless networks. We will summarize this algorithm through an example in 2-D space. So, let points $\{\mathbf{p_i}, \mathbf{p_j}, \mathbf{p_k}, \mathbf{p_l}\} \in \mathbf{P}$ be the locations of some nodes in a network. Then, we can calculate the magnitude of barycentric coordinates of point l as:

$$|\boldsymbol{\lambda}_i^{(l)}| = \sqrt{\frac{D(\mathbf{p_l},\mathbf{p_j},\mathbf{p_k})}{D(\mathbf{p_i},\mathbf{p_j},\mathbf{p_k})}} \quad |\boldsymbol{\lambda}_j^{(l)}| = \sqrt{\frac{D(\mathbf{p_i},\mathbf{p_l},\mathbf{p_k})}{D(\mathbf{p_i},\mathbf{p_j},\mathbf{p_k})}} \quad |\boldsymbol{\lambda}_k^{(l)}| = \sqrt{\frac{D(\mathbf{p_i},\mathbf{p_j},\mathbf{p_l})}{D(\mathbf{p_i},\mathbf{p_j},\mathbf{p_k})}} \tag{3.4}$$

Now, by definition of barycentric networks in Definition 2, their sum must be equal to one. Therefore, if we suppose that node l is inside the convex-hull of the other nodes, then the sum of its barycentric coordinates magnitudes should also sum to one. This acts as a inclusion test to verify if nodes are inside or outside the convex-hull of its neighbors.

Definition 4. *Let the tuple $\mathcal{P} = (\mathbf{p_0}, \cdots, \mathbf{p_d})$ be a frame for an affine space $\mathbf{P}$. Then, given* *a point* $\mathbf{p} \in \mathbf{P}$*,* $\mathbf{p}$ *is inside the convex-hull of* $\mathcal{P}$ *if and only if*

$$\sum_{i=1}^{n} |\boldsymbol{\lambda}_i^{(l)}| = 1. \tag{3.5}$$

Where, $|\boldsymbol{\lambda}_i^{(l)}| = 0$ *if* $i \notin \mathcal{P}$*. We call this the inclusion test.*

Therefore, if a node passes the inclusion test for a subset of its neighbors, it can be correctly localize according to the following set of equations which define DILOC:

$$\begin{cases} \mathbf{p^{(l,\,t+1)}} = \sum_{i=1}^{n} \boldsymbol{\lambda}_i^{(l)} \mathbf{p^{(i,\,t)}} & \text{if } l \text{ is unknown,} \\ \mathbf{p^{(l,\,t+1)}} = \mathbf{p^{(l,\,t)}} & \text{if } l \text{ is anchor.} \end{cases} \tag{3.6}$$

The main advantage of DILOC's algorithm is that it can be written as an Absorbing Markov process, which allows one to use its theory to prove the method's convergence and facilitates its usability in many other setups such as [48, 49]. Next, we define ECHO which was proposed by Diao *et al.* in [33] as an extension of DILOC.

3.2.2. ECHO

This algorithm differentiates itself from DILOC in two major ways. First, it proposes a way to compute the correct barycentric coordinate signs in the two dimensional case. This allows one to utilize any possible subset of neighbors of a node to compute its barycentric coordinates. Second, it defines the generalized barycentric coordinates which are then used to form a linear system able to solve the localization problem under some assumptions. Diao *et al.* in [33], proposes Algorithm 1 to compute the correct signs for barycentric coordinates in the two dimensional case. The algorithm is composed of a series of if and else statements, which dissects all possible node arrangements. Moreover, all of these checks must be done for each possible neighboring node combination of $d + 1$ nodes for each node in the network. This is done in order to compute, what they called the Generalized Barycentric Coordinates, which we present in Definition 5.

Definition 5 (Generalized Barycentric Coordinate [33].)**.** *For each node* l*, its generalized*

Algorithm 1 Signs of barycentric coordinates in 2D.

```
Input: $|\boldsymbol{\lambda}_i^{(l)}|$, $|\boldsymbol{\lambda}_j^{(l)}|$, $|\boldsymbol{\lambda}_k^{(l)}|$, $Z_{li}$, $Z_{lj}$, $Z_{lk}$, $Z_{ij}$, $Z_{ik}$, $Z_{jk}$
Output: $\sigma_i^{(l)}$, $\sigma_j^{(l)}$, $\sigma_k^{(l)}$
  if $\sigma_i^{(l)}|\boldsymbol{\lambda}_i^{(l)}| + \sigma_j^{(l)}|\boldsymbol{\lambda}_j^{(l)}| + \sigma_k^{(l)}|\boldsymbol{\lambda}_k^{(l)}| = 1$ has unique solution then
    return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)})$
  else if $|\boldsymbol{\lambda}_i^{(l)}| == 0$ or $|\boldsymbol{\lambda}_j^{(l)}| == 0$ or $|\boldsymbol{\lambda}_k^{(l)}| == 0$ then
    if $|\boldsymbol{\lambda}_j^{(l)}| \leq 1$ and $|\boldsymbol{\lambda}_k^{(l)}| \leq 1$ then
      return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (1, 1, 1)$
    else if $|\boldsymbol{\lambda}_j^{(l)}| > 1$ and $|\boldsymbol{\lambda}_j^{(l)}| > |\boldsymbol{\lambda}_k^{(l)}|$ then
      return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (1, 1, -1)$
    else if $|\boldsymbol{\lambda}_k^{(l)}| > 1$ and $|\boldsymbol{\lambda}_k^{(l)}| > |\boldsymbol{\lambda}_l^{(l)}|$ then
      return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (1, -1, 1)$
    end if
  else if $Z_{lj} == Z_{ik}$ and $Z_{lk} == Z_{ij}$ and $Z_{li}^2 == 2Z_{ij}^2 + 2Z_{ik}^2 - Z_{jk}^2$ then
    return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (-1, 1, 1)$
  else if $Z_{lj} < Z_{ij}^2 + Z_{li}^2$ then
    return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (1, 1, -1)$
  else if $Z_{lj} > Z_{ij}^2 + Z_{li}^2$ then
    return $(\sigma_i^{(l)}, \sigma_j^{(l)}, \sigma_k^{(l)}) = (1, -1, 1)$
  end if
```

barycentric coordinate, $\boldsymbol{\lambda}_l \in \mathbb{R}^m$, can be computed as follows

$$\boldsymbol{\lambda}_l^T = \frac{1}{|\mathcal{I}^{(l)}|} \sum_{i=1}^{|\mathcal{I}^{(l)}|} L_{ij}^{(l)}, \tag{3.7}$$

where $L^{(l)}$ is the matrix formed by concatenating all possible barycentric coordinates $\boldsymbol{\lambda}^T$ which are computed using both Definition 3 and Algorithm 1.

As the generalized barycentric coordinates in equation (3.7) are computed through averaging, the property of summing to one is preserved. These new set of coordinates are then used to compose a linear system of equations that relates each generalized barycentric coordinate to the set of Cartesian coordinates which represent the node locations. This results in a system with n equations in $u < n$ variables, which can be solved under the assumption, that each node has at least $d + 1$ distinct paths to the anchor nodes on the network, by isolating a subset of equations as described in [33]. We proceed in a similar form in Section 3.3.4, but with a different set of equations.

3.3. Our approach

Our approach distinguishes itself from the other existing literature as we first provide a novel and formal way to compute barycentric coordinates for any and every point $\mathbf{p} \in \mathbf{P}$ in closed form; second, utilizing our formulation of barycentric coordinates, we extend the case where one allows arbitrary placement of anchors among unknowns to d-dimensions, and finally; third, we provide a noiseless centralized algorithm as well as a noisy distributed algorithm able to solve our proposed localization problem in Chapter 2. Let's start by defining Cayley-Menger bi-determinants in the next section.

3.3.1. Cayley-Menger bi-determinants

While Cayley-Menger determinants work on a single set of points in a d-dimensional space; we introduce the concept of Cayley-Menger bi-determinants, also defined in [8] without any formal proof. Cayley-Menger bi-determinants operate on two sets of points; providing a relation between the product of volumes of each set and the Euclidean distances between points of different sets.

Definition 6. *Let two tuples of $d+1$ points, $\mathcal{P} = (\mathbf{p_0}, \ldots, \mathbf{p_d})$ and $\mathcal{Q} = (\mathbf{q_0}, \ldots, \mathbf{q_d})$, be defined by their Cartesian coordinates, such that $\mathbf{p_i}$ and $\mathbf{q_i} \in \mathbb{R}^d$ for $0 \le i \le d$. Then, the Cayley-Menger bi-determinant of $\mathcal{P}$ and $\mathcal{Q}$ is defined as follows:*

$$D(\mathcal{P};\mathcal{Q}) = D(\mathbf{p_0}, \ldots, \mathbf{p_d}; \mathbf{q_0}, \ldots, \mathbf{q_d}) =$$
$$2\left(-\frac{1}{2}\right)^{d+1} \begin{vmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & d(\mathbf{p_0},\mathbf{q_0})^2 & d(\mathbf{p_0},\mathbf{q_1})^2 & \cdots & d(\mathbf{p_0},\mathbf{q_d})^2 \\ 1 & d(\mathbf{p_1},\mathbf{q_0})^2 & d(\mathbf{p_1},\mathbf{q_1})^2 & \cdots & d(\mathbf{p_1},\mathbf{q_d})^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d(\mathbf{p_d},\mathbf{q_0})^2 & d(\mathbf{p_d},\mathbf{q_1})^2 & \cdots & d(\mathbf{p_d},\mathbf{q_d})^2 \end{vmatrix}. \tag{3.8}$$

In Theorem 2, we formally specify the relationship between signed volumes of tuples, $\mathcal{P}$ and $\mathcal{Q}$, in d-dimensional Euclidean space and their Cayley-Menger bi-determinant.

Theorem 2. *The Cayley-Menger bi-determinant of two tuples of $d+1$ points in $\mathbb{R}^d$, $\mathcal{P} = (\mathbf{p_0}, \ldots, \mathbf{p_d})$ and $\mathcal{Q} = (\mathbf{q_0}, \ldots, \mathbf{q_d})$, is related to the products of the signed volumes of each independent tuple, by*

$$D(\mathcal{P};\mathcal{Q}) = (d!)\,\textit{Vol}(\mathcal{P}) \cdot (d!)\,\textit{Vol}(\mathcal{Q}). \tag{3.9}$$

Proof. See Appendix A.1. □

Corollary 1. *The Cayley-Menger determinant of the tuple of $d+1$ points $\mathcal{P}$ can be defined from the bi-determinant as follows:*

$$D(\mathcal{P}) = D(\mathbf{p_0}, \ldots, \mathbf{p_d}) = D(\mathbf{p_0}, \ldots, \mathbf{p_d}; \mathbf{p_0}, \ldots, \mathbf{p_d}). \tag{3.10}$$

Proof. See Appendix A.1 by taking $\mathcal{Q} = \mathcal{P}$. □

It is very important to notice that determinants in general are alternating forms, so the order in which its elements are arranged is essential to its correct computation. This becomes specially important to bi-determinants as changing the order of points may incur in a change of signal. Meanwhile, it is not as important to preserve order for Cayley-Menger determinants as we are only interested in the magnitude of barycentric coordinates in this case.

The following sections will show how we leverage Theorem 2 in order to compute barycentric coordinates of points inside and outside the convex-hull of its $d+1$ neighbors, which allow us to obtain similar, but more concise results than [33].

3.3.2. Computing barycentric coordinates

Theorem 3. *Let the tuple $\mathcal{P} = (\mathbf{p_0}, \cdots, \mathbf{p_d})$ be a frame for an affine space $\mathbf{P}$. Then, any point $\mathbf{p} \in \mathbf{P}$ can be localized in relation to the tuple $\mathcal{P}$ by its barycentric coordinates $\boldsymbol{\lambda}$ as* *follows:*

$$\boldsymbol{\lambda_i} = \frac{D(\mathbf{p_0},\ldots,\mathbf{p_d};\mathbf{p_0},\ldots,\mathbf{p_{i-1}},\mathbf{p},\mathbf{p_{i+1}},\ldots,\mathbf{p_d})}{D(\mathbf{p_0},\ldots,\mathbf{p_d})} \tag{3.11}$$

Proof. From Definition 2, the following relations hold:

$$\mathbf{p} = \textstyle\sum_i \lambda_i \mathbf{p_i}, \quad \sum_i \lambda_i = 1 \quad \Leftrightarrow \quad \begin{bmatrix} P \\ \mathbb{1}^T \end{bmatrix} \lambda = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}. \tag{3.12}$$

Where $P = [\mathbf{p_0},\mathbf{p_1},\cdots,\mathbf{p_d}]$. Now, suppose our set of nodes form a frame in an affine space, then the solution is unique by Definition 2. The latter implies that $V_P^T = [P^T\ \mathbb{1}]$ has non-zero determinant, which allows us to use Cramer's Rule to solve the linear system (3.12). Therefore, by defining $\widehat{P_i}$ as the matrix obtained from P by replacing its i^{th} column with $\mathbf{p}$, *i. e.* $\widehat{P_i} = [\mathbf{p_0},\cdots,\mathbf{p_{i-1}},\mathbf{p},\mathbf{p_{i+1}},\cdots,\mathbf{p_d}]$, we can compute each barycentric coordinate, $\boldsymbol{\lambda_i}$, as follows

$$\boldsymbol{\lambda_i} = \begin{vmatrix} \widehat{P_i} \\ \mathbb{1}^T \end{vmatrix} \Big/ \begin{vmatrix} P \\ \mathbb{1}^T \end{vmatrix}. \tag{3.13}$$

Meanwhile, the volume of the sets of points $\mathcal{P}$ and $\widehat{\mathcal{P}_i}$ are given by

$$\text{Vol}(\mathcal{P}) \;=\; \tfrac{1}{d!}\begin{vmatrix} P \\ \mathbb{1}^T \end{vmatrix}, \qquad \text{Vol}(\widehat{\mathcal{P}_i}) \;=\; \tfrac{1}{d!}\begin{vmatrix} \widehat{P_i} \\ \mathbb{1}^T \end{vmatrix}. \tag{3.14}$$

Then, each barycentric coordinates, $\boldsymbol{\lambda_i}$, can be computed in terms of these volumes by

$$\boldsymbol{\lambda_i} = \frac{\begin{vmatrix} \widehat{P_i} \\ \mathbb{1}^T \end{vmatrix}}{\begin{vmatrix} P \\ \mathbb{1}^T \end{vmatrix}} = \frac{(d!)\text{Vol}(\widehat{\mathcal{P}_i})}{(d!)\text{Vol}(\mathcal{P})}. \tag{3.15}$$

Now, one may see that the following multiplication by ”one” becomes a neat mathematical trick.

$$\boldsymbol{\lambda_i} = \frac{(d!)\mathsf{Vol}(\mathcal{P})}{(d!)\mathsf{Vol}(\mathcal{P})}\frac{(d!)\mathsf{Vol}(\widehat{\mathcal{P}_i})}{(d!)\mathsf{Vol}(\mathcal{P})} \tag{3.16}$$

Substituting numerator by its equivalent to Theorem 1 and denominator by Corollary 1, we arrive to our proposed solution. Therefore, we conclude that the barycentric coordinates, $\boldsymbol{\lambda_i}$ for all $0 \leq i \leq d$, can be computed using Cayley-Menger bi-determinants and determinants. □

Instead of using equation (3.16), previous works [32–34] would use the following relation (3.17), which is based on (3.15), to compute absolute values of barycentric coordinates.

$$\boldsymbol{\lambda_i}^2 = \frac{(d!)^2\mathsf{Vol}(\widehat{\mathcal{P}_i})^2}{(d!)^2\mathsf{Vol}(\mathcal{P})^2} = \frac{D(\widehat{\mathcal{P}_i})}{D(\mathcal{P})}. \tag{3.17}$$

While (3.17) can provide magnitudes of barycentric coordinates of a point in relation to an affine frame, it can not provide the correct sign. This restriction did not impede the development of DILOC's theory, as it is restricted to points inside the convex-hull of anchors (strictly positive) as we saw before in Section 3.2.1. Contrarily, it became a restriction for the full development of ECHO, as they had to come up with many geometric rules to determine correct signs even in 2D, not providing the same for 3D nor any larger dimensional case, as seen in Section 3.2.2. On the other hand, we emphasize that our method, given by Theorem 3, is able to compute the full barycentric coordinates, magnitude and signs, given the same conditions.

Notice that each point's barycentric coordinate sign is associated to the overall position of this point in relation to the ones forming the affine frame used in its computation. Any point strictly inside the convex-hull of the affine frame will have strictly positive coordinates, otherwise it will have at least one zero or negative coordinate. It is impossible to have all non-zero barycentric coordinates with negative signs. Moreover, the order in which the different signs are given depends on the partial ordering of $d+1$ hyper-planes in $d-1$ dimensions generated on d-dimensional space.

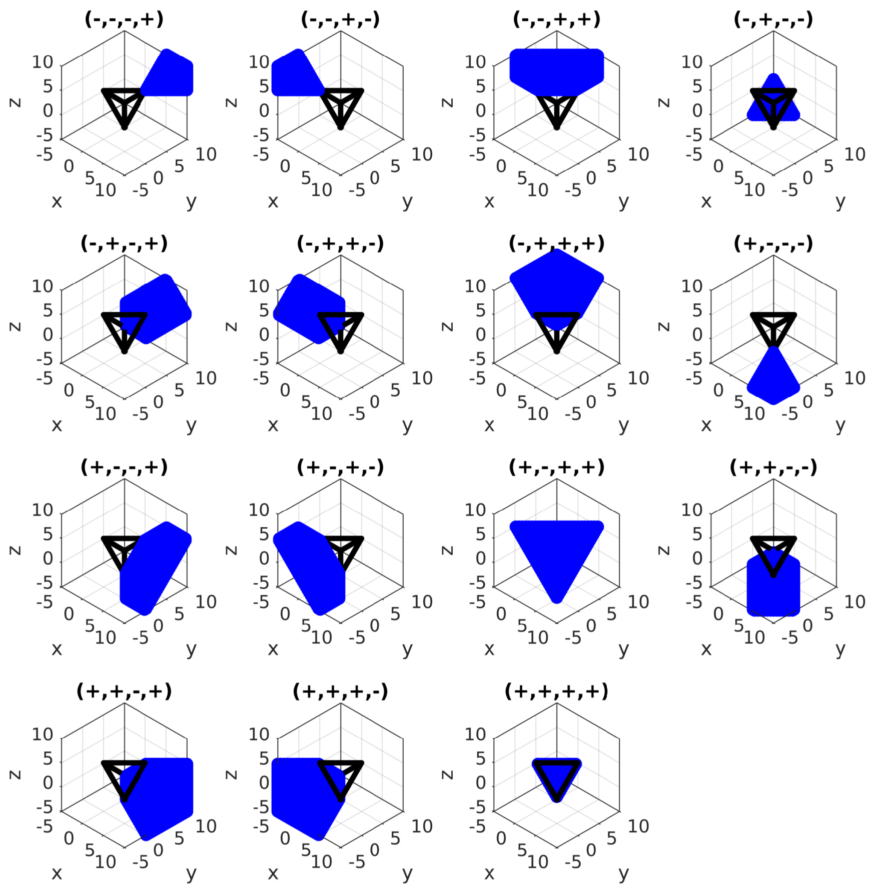


Figure 5: Regions of different Barycentric Coordinate signs with non-zero elements. Here, we consider only regions were coordinates are either strictly positive or strictly negative. The black line segments define the boundaries of the chosen points convex-hull.

It is important to understand that if we consider each node as a point in an affine space, then allowing nodes to be outside of the convex-hull of its neighbors, is equivalent of having zero or negative barycentric coordinates. This constitutes the main difference between DILOC's and ECHO's algorithm, [32] and [33] respectively.

Fig. 5 shows an example in three-dimensional space of the $2^{d+1} - 1$ possible regions where the barycentric coordinate signs are strictly positive or negative. The convex-hull of the set of $d + 1$ points forming the affine frame is depicted by having all its edges blacked in the figure. Meanwhile, Fig. 6 shows some of the possible regions when we consider coordinates which are striclty zero. Notice, that we can not show all possible regions in this

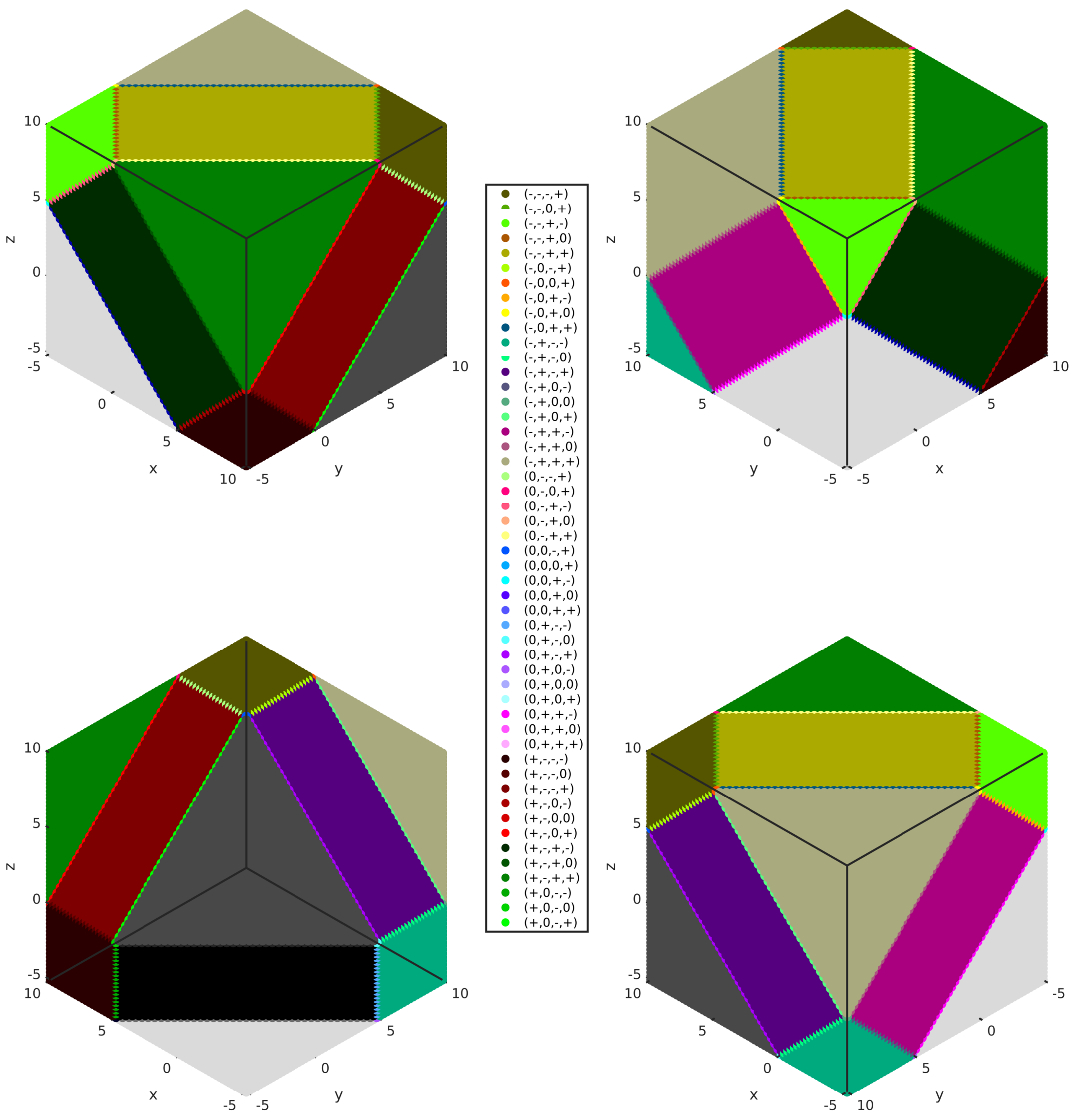


Figure 6: Regions of different Barycentric Coordinate signs with zero elements. In this figure, we specify some, but not all, possible regions in space diving them by strictly positive, strictly negative or strictly zero coordinates.

case, as some are hidden beneath others and some are plane segments or even points.

3.3.3. Computing barycentric coordinates for every node l

In the previous two sections, we defined Cayley-Menger bi-determinants and barycentric coordinates which we will apply in order to solve our localization problem. First, we define that each network node can have its location modelled as a point in an Euclidean and

Affine Space. Therefore, it can have Cartesian and barycentric coordinates.

The easiest localization problem occurs when all unknown nodes are directly connected to all anchors and the anchors form an affine frame. In this case, each node barycentric coordinate can be computed from measuring their relative distance with the anchors. Therefore, by using the previous results we arrive at each node Cartesian coordinates.

In the case where one unknown node is not directly connected to all anchors, then we must seek a combination of its neighbors that form an affine frame and are themselves localizable based on their neighbors. This principle is applied on the standard Trilateration method in reverse. From anchors, localize first degree neighbors, then second degree and so on and so forth.

As we wish to arrive to a solution similar to the ones presented in [32, 33], we need to find all necessary barycentric coordinates for each unknown node in a way that our system of equations has a unique solution, *i.e.* all nodes become localizable at the same time. This is not a trivial problem as the total amount of possible barycentric coordinates for each node is a combination of its number of neighbors and $d+1$. Thus, for large and dense networks, the number of possible choices can explode.

In [33], Diao *et al.* proposed a solution for this problem by computing a generalized barycentric coordinate which they defined from the average of all possible barycentric coordinates of a node in a network. Next, we will propose a different formulation also using all possible barycentric coordinates.

Let $\mathcal{N}^{(l)}$ be the index set of neighbors of node l, defined as follows

$$\mathcal{N}^{(l)} = \{j \in \mathcal{V} \setminus \{l\} \,|\, (l, j) \in \mathcal{E}\} \,. \tag{3.18}$$

Let $\mathcal{I}^{(l)}$ be a family of sets of $d+1$ indexes given by the combination without repetition of members of $\mathcal{N}^{(l)}$, which are also neighbors from one another.

$$\mathcal{I}^{(l)} = \{\mathcal{V}_j \subset \mathcal{N}^{(l)^{\times d+1}} | \mathcal{J}_j = \{\mathcal{V}_j, \mathcal{E}_j\} \in \mathbb{K}_{d+1}, \mathcal{E}_j \subset \mathcal{E}\} \tag{3.19}$$

Where, $\mathcal{J}_j$ is a subgraph of our network graph $\mathcal{G}$ and $\mathbb{K}_{d+1}$ is the set of complete graphs with $d+1$ vertices. Therefore, the cardinality of $\mathcal{I}^{(l)}$ is $|\mathcal{I}^{(l)}| \leq \begin{pmatrix} |\mathcal{N}^{(l)}| \\ d+1 \end{pmatrix}$.

For each node l, we define tuples of $d+1$ points in $\mathbb{R}^d$, $\mathcal{X}_i^{(l)}$, $1 \leq i \leq |\mathcal{I}^{(l)}|$, such that

$$\mathcal{X}_i^{(l)} = \{\mathbf{x}_{\mathcal{I}_{i1}^{(l)}}, \mathbf{x}_{\mathcal{I}_{i2}^{(l)}}, \cdots, \mathbf{x}_{\mathcal{I}_{id+1}^{(l)}}\}, \tag{3.20}$$

and sets of $d+1$ points in the same space, $\widehat{\mathcal{X}}_i^{(l)}$, $1 \leq j \leq d+1$, such that

$$\widehat{\mathcal{X}}_{i_j}^{(l)} = \{\mathbf{x}_{\mathcal{I}_{i1}^{(l)}}, \cdots, \mathbf{x}_{\mathcal{I}_{ij-1}^{(l)}}, \mathbf{x}_l, \mathbf{x}_{\mathcal{I}_{ij+1}^{(l)}}, \cdots, \mathbf{x}_{\mathcal{I}_{id+1}^{(l)}}\}. \tag{3.21}$$

Using the previously defined sets with the correct indexes applied to equation (3.16), one can compute all possible barycentric coordinates for each node l.

Theorem 4. *For each node l and each set $\mathcal{I}_i^{(l)}$, if $D(\mathcal{X}_i^{(l)}) \neq 0, \forall i$, then all possible barycentric coordinates of node l, with respect to its neighbors identified by the set $\mathcal{I}_i^{(l)}$, can be arranged in matrix form $L^{(l)} \in \mathbb{R}^{|\mathcal{I}^{(l)}| \times n}$*

$$L_{ik}^{(l)} = \begin{cases} \dfrac{D\left(\mathcal{X}_i^{(l)}; \widehat{\mathcal{X}}_{i_j}^{(l)}\right)}{D\left(\mathcal{X}_i^{(l)}\right)}, & \textit{if } k = \mathcal{I}_{ij}^{(l)} \\ 0, & \textit{otherwise} \end{cases}. \tag{3.22}$$

Proof. Follows from equation (3.16), by taking the appropriate nodes and their indexes (3.20) and (3.21). □

In the next section, we utilized Theorem 4 to solve our proposed problem in a noiseless environment using a centralized algorithm.

3.3.4. Generalizing barycentric coordinates: The centralized solution

By defining the matrix $X \in \mathbb{R}^{n \times d}$, such that $X^T = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$, the previous Theorem 4 thus states that

$$L^{(l)} X = \mathbb{1} \cdot \mathbf{x_l}^T. \tag{3.23}$$

One possible way to utilize the previous result in Theorem 4 and equation (3.23) is to compute the Generalized Barycentric Coordinates proposed by Diao et. al. in [33]. These generalized coordinates are computed by averaging all possible barycentric coordinates for each node l as stated in Definition 5. One can use all n generalized barycentric coordinates in order to compute the unknown nodes coordinates by simply solving a centralized linear system as proposed in [33].

As a curiosity, we show here how to obtain the generalized barycentric coordinates, $\boldsymbol{\lambda}_l$, from matrix $L^{(l)}$. Starting from equation (3.23), we compute the averaging process as follows, where $\mathbb{1} \in \mathbb{R}^{|\mathcal{I}_l|}$,

$$\mathbb{1}^T L^{(l)} X = \mathbb{1}^T \mathbb{1} \cdot \mathbf{x_l}^T \tag{3.24}$$

$$\frac{1}{|\mathcal{I}^{(l)}|} \mathbb{1}^T L^{(l)} X = \mathbf{x_l}^T \tag{3.25}$$

$$\boldsymbol{\lambda}_l^T X = \mathbf{x_l}^T \tag{3.26}$$

$$\boldsymbol{\lambda}_l = \frac{1}{|\mathcal{I}^{(l)}|} \mathbb{1}^T L^{(l)} \tag{3.27}$$

Despite the apparent simplicity of the previous solution, it does not conform with the generic optimization problem given in [47], which we wish to utilize in order to solve our problem with noisy measurements in a distributed manner. Therefore, we seek an alternative manipulation of equation (3.23).

Proposition 1 (Generalized Barycentric Matrix)**.** *For each node l, its generalized barycen-*

tric matrix, $\mathbb{L}^{(l)} \in \mathbb{R}^{|\mathcal{I}^{(l)}| \times n}$, is given by:

$$\mathbb{L}^{(l)} = L^{(l)} - \mathbb{1}\mathbf{e_l}^T. \tag{3.28}$$

Proof. From the manipulation of equation (3.23) we have, where $\mathbb{1} \in \mathbb{R}^{|\mathcal{I}^{(l)}|}$,

$$L^{(l)} X = \mathbb{1} \cdot \mathbf{x_l}^T \tag{3.29}$$

$$\left(L^{(l)} - \mathbb{1}\mathbf{e_l}^T\right) X = 0 \tag{3.30}$$

$$\mathbb{L}^{(l)} X = 0. \tag{3.31}$$

□

In order to use Proposition 1 to solve our localization problem, one may concatenate matrices $\mathbb{L}^{(l)}$ and solve the following set of equations

$$\begin{bmatrix} \mathbb{L}^{(1)} \\ \mathbb{L}^{(2)} \\ \vdots \\ \mathbb{L}^{(n)} \end{bmatrix} X = 0 \tag{3.32}$$

which relates to computing the kernel of the rectangular matrix that defines our over-determined linear system.

Theorem 5. *The localization problem proposed can be solved through the solution of the linear system given by* (3.32). *Its centralized solution for the unknown nodes is:*

$$X_{\mathcal{U}} = -\left(\sum_{l=1}^{n} \mathbb{L}_{:\mathcal{U}}^{(l)^T} \mathbb{L}_{:\mathcal{U}}^{(l)}\right)^{-1} \left(\sum_{l=1}^{n} \mathbb{L}_{:\mathcal{U}}^{(l)^T} \mathbb{L}_{:\mathcal{A}}^{(l)}\right) X_{\mathcal{A}}, \tag{3.33}$$

where $\mathbb{L}_{:\mathcal{A}}^{(l)}$ and $\mathbb{L}_{:\mathcal{U}}^{(l)}$ are the sub-matrices obtained by selecting columns indexed by anchor and unknown nodes respectively and if there exist the inverse of matrix $\sum_{l=1}^{n} \mathbb{L}_{:\mathcal{U}}^{(l)^T} \mathbb{L}_{:\mathcal{U}}^{(l)}$.

If this inverse does not exits, one may compute a least squares estimate by solving the following equation:

$$\begin{aligned} X_{\mathcal{U}} \;=\; & -\left\{\left(\sum_{l=1}^{n}\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\\ \mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\end{bmatrix}\right)^T\left(\sum_{l=1}^{n}\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\\ \mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\end{bmatrix}\right)\right\}^{-1}\cdot \\ & \cdot\left(\sum_{l=1}^{n}\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\\ \mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\end{bmatrix}\right)^T\left(\sum_{l=1}^{n}\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{A}}^{(l)}\\ \mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{A}}^{(l)}\end{bmatrix}\right)X_{\mathcal{A}}, \end{aligned} \tag{3.34}$$

Proof. We approach this problem, by transforming using the least squares approach to the system of equations in (3.32):

$$\begin{bmatrix}\mathbb{L}^{(1)}\\ \mathbb{L}^{(2)}\\ \vdots\\ \mathbb{L}^{(n)}\end{bmatrix}^T\begin{bmatrix}\mathbb{L}^{(1)}\\ \mathbb{L}^{(2)}\\ \vdots\\ \mathbb{L}^{(n)}\end{bmatrix}X = 0 \tag{3.35}$$

$$\sum_{l=1}^{n}\left(\mathbb{L}^{(l)^T}\mathbb{L}^{(l)}\right)X = 0 \tag{3.36}$$

Now, because some elements of matrix X are previously known, the anchor coordinates $X_{\mathcal{A}}$, then we will re-arrange our previous matrices $\mathbb{L}$ by applying row and column permutations so that we have:

$$\sum_{l=1}^{n}\left(\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)^T}\\ \mathbb{L}_{:\mathcal{U}}^{(l)^T}\end{bmatrix}\begin{bmatrix}\mathbb{L}_{:\mathcal{A}}^{(l)} & \mathbb{L}_{:\mathcal{U}}^{(l)}\end{bmatrix}\right)\begin{bmatrix}X_{\mathcal{A}}\\ X_{\mathcal{U}}\end{bmatrix} = 0 \tag{3.37}$$

$$\begin{aligned} \sum_{l=1}^{n}\left(\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{A}}^{(l)}\right)X_{\mathcal{A}} + \sum_{l=1}^{n}\left(\mathbb{L}_{:\mathcal{A}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\right)X_{\mathcal{U}} &= 0 \\ \sum_{l=1}^{n}\left(\mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{A}}^{(l)}\right)X_{\mathcal{A}} + \sum_{l=1}^{n}\left(\mathbb{L}_{:\mathcal{U}}^{(l)^T}\mathbb{L}_{:\mathcal{U}}^{(l)}\right)X_{\mathcal{U}} &= 0 \end{aligned} \tag{3.38}$$

Finally, similarly to the previous methods, DILOC and ECHO, we choose to solve the second part of the previous equation which results in our solution provided in (3.33) if the inverse of $\sum_{l=1}^{n} \mathbb{L}_{:\mathcal{U}}^{(l)^T} \mathbb{L}_{:\mathcal{U}}^{(l)}$ exists, otherwise we proceed as in equation (3.34). $\square$

3.4. Algorithm complexity

The number of barycentric coordinates needed to be computed is given by the cardinality of the family sets, $\mathcal{I}^{(l)}$, from equation (3.19). Because of that, an implementation of our algorithm will have worst case scenario for its computational complexity whenever its network graph $\mathcal{G}$ is complete, i.e. all nodes are inter-connected. In which case, the cardinality for all nodes is equal to $\begin{pmatrix} n \\ d+1 \end{pmatrix}$. Therefore, in order to compute matrix $\mathbb{L}$, the complexity of our algorithm becomes $O\left(d^4 n \begin{pmatrix} n \\ d+1 \end{pmatrix}\right)$, where n is the number of nodes and d is the dimension of each node. Therefore, the centralized case in which we find the final solution by inverting a block matrix derived from $\mathbb{L}$, operation which has $O(n^3)$, has final complexity of $O\left(d^4 n \begin{pmatrix} n \\ d+1 \end{pmatrix} + n^3\right)$.

CHAPTER 4 : A NOISY DISTRIBUTED SOLUTION

In this chapter we deal with two main extensions of our centralized noiseless algorithm which was proposed before in Chapter 3, which are first to enable our algorithm to accept zero mean additive random noise to each range measurement and second to transform its computation from centralized to distributed. We tackle the former by leveraging the idea of averaging twice, one over the range measurements inputs and the other over the already computed barycentric coordinates. It is easy to understand why we apply averages to noisy range measurements with zero mean additive random noise as by the Law of Large Numbers, the average will converge to the noiseless value given a sufficiently high amount of samples. But, one may not say the same for the barycentric coordinates as they are computed via determinants of squared averaged noisy range values, which is a non linear operation, so one can not apply standard probability distributions to barycentric coordinates.

We apply the second averaging process with the intention of mitigating values discrepancies between barycentric coordinates computed from one iteration to another. Such discrepancies may occur because of the non linearity of the Cayley-Menger computation or even because of numerical error from values near zero on the denominator of the barycentric coordinates computation. Moreover, it is important that one checks for these simplices with near zero volume and restrict their usability. We checked the efficacy of this double averaging process utilizing simulations in Matlab considering both the solution of the centralized case with averaged values at each time or a derived gradient descent algorithm, formally shown in Section 4.2. The simulated results may be seen at Section 5.3.

For the second extension, which is making our algorithm distributed, we have leveraged an existing result by Shahrampour and Jadbabaie [47] on an optimization algorithm for networks with cost functions varying over a time horizon. From our point of view, this algorithm was developed using as basis the Bregman Divergence and a Distributed Consensus Algorithm. It enables us to obtain iteratively better location estimates over time as our algorithm becomes a modified gradient descent algorithm with convex cost functions. This happens because we utilize the Euclidean distance in order to calculate the Bregman Divergence.

In the following sections, we will formally introduce the steps taken to arrive to our distributed noisy algorithm for the d-dimensional localization problem in wireless networks with arbitrary anchor placement.

4.1. Averaging processes

Sec. 3.3.3 and 3.3.4 showed that the entries of the matrix $\mathbb{L}$ in (3.32), which specifies the linear system of equations for all nodes, have a nonlinear dependence on the range measurements due to the Cayley-Menger bi-determinant computations. In this section, we consider the relative localization problem in the presence of noisy measurements. The main challenge is that the distribution of the measurement noise $\delta_{ij}(t)$ is, in general, unknown and even if we were to make an assumption on its class (e.g., common choices include Gaussian, Laplace, Rayleigh, or Rice), when it is propagated through the nonlinear barycentric coordinate function in Theorem 4, the posterior would not be a standard or stable distribution. We begin by introducing two averaging schemes to mitigate the noise effects and allow Theorem 5 to be applied in the noisy case.

We follow the approach of Khan *et al.* [34] to deal with this challenge. Their idea is to leverage the Law of Large Numbers, which ensures that the iid measurements $[Z_t]_{ij}$, when averaged over time, almost surely converge to the noiseless range measurement $d(\mathbf{x}_i, \mathbf{x}_j)$, and compute the matrix $\mathbb{L}^{(l)}$ using averaged measurements. In detail, instead of computing $\mathbb{L}^{(l,t)}$ at each time step t using the raw measurements $\{[Z_t]_{ij} \mid (i,j) \in \mathcal{E}\}$, we compute $\mathbb{L}^{(l,t)}$ using the following *averaged range measurements*:

$$[\overline{Z}_t]_{ij} = \frac{t-1}{t}[\overline{Z}_{t-1}]_{ij} + \frac{1}{t}[Z_t]_{ij}, \text{ with } [\overline{Z}_0]_{ij} = 0. \tag{4.1}$$

This process, guarantees that after sufficient time passes, $\mathbb{L}^{(l,t)}$ will be close to the matrix resulting from the true node barycentric coordinates, and the noisy relative localization problem can again be solved via Theorem 5. Despite averaging measurement, there may be instances in which large instantaneous noise at time t or numerical errors from near zero volumes would lead to an inaccurate $\mathbb{L}^{(l,t)}$. To mitigate the noise introduced by the most recent measurement, we introduce a second averaging process over the elements of matrix $\mathbb{L}^{(l,t)}$. In detail, we propose the following steps for computing time-averaged blocks of $\mathbb{L}^{(l,t)}$:

$$\begin{aligned}\overline{\mathbb{L}}^{(l,t)}_{:\mathcal{A}} &= \frac{t-1}{t}\overline{\mathbb{L}}^{(l,t-1)}_{:\mathcal{A}} + \frac{1}{t}\mathbb{L}^{(l,t)}_{:\mathcal{A}} \\ \overline{\mathbb{L}}^{(l,t)}_{:\mathcal{U}} &= \frac{t-1}{t}\overline{\mathbb{L}}^{(l,t-1)}_{:\mathcal{U}} + \frac{1}{t}\mathbb{L}^{(l,t)}_{:\mathcal{U}}\end{aligned} \tag{4.2}$$

where $\overline{\mathbb{L}}^{(l,0)}_{:\mathcal{A}} = \mathbf{0} \in \mathbb{R}^{|\mathcal{I}^{(l)}|\times a}$ and $\overline{\mathbb{L}}^{(l,0)}_{:\mathcal{U}} = \mathbf{0} \in \mathbb{R}^{|\mathcal{I}^{(l)}|\times u}$. Given these definitions, we can return to the approach in Proposition 5 but this time using the time-averaged matrices. Leading to the following least-squares estimate of the unknown node locations:

$$\tilde{X}^{(t)}_{\mathcal{U}} = -\left(\sum_{l=1}^{n} \overline{\mathbb{L}}^{(l,t)^T}_{:\mathcal{U}} \overline{\mathbb{L}}^{(l,t)}_{:\mathcal{U}}\right)^{-1} \left(\sum_{l=1}^{n} \overline{\mathbb{L}}^{(l,t)^T}_{:\mathcal{U}} \overline{\mathbb{L}}^{(l,t)}_{:\mathcal{A}}\right) X_{\mathcal{A}}, \tag{4.3}$$

While this process mitigates the effect of measurement noise on the computation of blocks of $\mathbb{L}^{(l,t)}$ for large enough t, this computation requires centralized information. As a centralized solution to the relative localization problem is not suitable for large networks, we will in Sec. 4.3 develop a distributed gradient descent algorithm. To motivate the distributed algorithm, we first derive a gradient descent approach for the centralized case with noisy measurements.

4.2. Centralized gradient descent

Besides the proposed solution in Theorem 5, one may also apply optimization theory over the second equation in (3.38) to obtain the same solution. We are emphasizing this point as it serves as the starting point for our distributed algorithm. Therefore,

Lemma 1. *The localization problem proposed can also be solved through the following* *optimization process:*

$$\tilde{X}_{\mathcal{U}}^{(t)} := \underset{X \in \mathbb{R}^{u \times d}}{\arg\min} \sum_{l=1}^{n} \left\| \begin{bmatrix} \overline{\mathbb{L}}_{:\mathcal{A}}^{(l,t)} & \overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)} \end{bmatrix} \begin{bmatrix} X_{\mathcal{A}} \\ X \end{bmatrix} \right\|_{fro}^{2} \tag{4.4}$$

where $\mathbb{L}_{:\mathcal{A}}^{(l)}$ and $\mathbb{L}_{:\mathcal{U}}^{(l)}$ are the sub-matrices obtained by selecting columns indexed by anchor and unknown nodes respectively.

The aforementioned minimization can be solved using the iterative centralized gradient descent algorithm for computing $\tilde{X}_{\mathcal{U}}$ as follows:

$$\tilde{X}_{\mathcal{U}}^{(t+1)} = \tilde{X}_{\mathcal{U}}^{(t)} - \eta^{(t)} G^{(t)}, \tag{4.5}$$

Where the gradient $G^{(t)}$ of the objective function in (4.4) defined in Lemma 1 is:

$$G^{(t)} := 2 \sum_{l=1}^{n} \overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)^T} \left(\overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)} X_{\mathcal{U}} + \overline{\mathbb{L}}_{\mathcal{A}}^{(l,t)} X_{\mathcal{A}} \right). \tag{4.6}$$

The gradient $G^{(t)}$ is evaluated at $\tilde{X}_{\mathcal{U}}^{(t)}$ and the step size $\eta^{(t)}$ determines the stability and convergence speed of the algorithm. One choice for $\eta^{(t)}$ that we found to work particularly well is based on the Barzilai-Borwein method [67]:

$$\eta^{(t)} := \frac{\mathbf{tr}\left(\left(\tilde{X}_{\mathcal{U}}^{(t)} - \tilde{X}_{\mathcal{U}}^{(t-1)} \right)^T \left(G^{(t)} - G^{(t-1)} \right) \right)}{\mathbf{tr}\left(\left(G^{(t)} - G^{(t-1)} \right)^T \left(G^{(t)} - G^{(t-1)} \right) \right)}.$$

This algorithm is initialized by computing $\tilde{X}_{\mathcal{U}}^{(1)}$ from the first set of range measurements via (3.33), this operation may lead to gross estimates that a far from the true node locations. In an attempt to improve this first estimate, we chose to utilize the centroid of a box containing the entire network as initial guess for estimates that are outside of such box.

4.3. Distributed Online Gradient Descent

In the presence of noisy measurements, the constraints in equation (3.32) can only be satisfied approximately. Hence, we use the same two-phase averaging process as in equation (4.2) in Section 4.1 to define $\overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)}$ and $\overline{\mathbb{L}}_{:\mathcal{A}}^{(l,t)}$ and aim to compute a least-squares estimation for X_u based on Lemma 1.

Notice that each matrix $\overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)}$ and $\overline{\mathbb{L}}_{:\mathcal{A}}^{(l,t)}$ can be computed locally in each node i with 2-hop information. But each neighbor already stores information about its neighbors. Therefore, all necessary computation can be done with only 1-hop communication. The structure of this interaction is defined with respect to a communication matrix W, selected based on Xiao *et al.* [46] to be

$$W = I - L_{\mathcal{G}} / \max\{eig(L_{\mathcal{G}})\} \tag{4.7}$$

where $L_{\mathcal{G}}$ is the Laplacian matrix of graph $\mathcal{G}$ and $eig(\cdot)$ computes the eigenvalues of the respective matrix. With these results one may directly utilize [47] algorithm with the Euclidean Distance as the Bregman Divergence function to arrive at the following theorem.

Theorem 6. *Given anchor node locations $\mathcal{X}_{\mathcal{A}} \subset \mathcal{X}$ of a static sensor network and noisy range measurements $[Z_t]_{ij} = d(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}^{(t)} \in \mathbb{R}$, $\forall (i,j) \in \mathcal{E}$ and $t = 0, \ldots, T$. We can estimate the locations of all unknown nodes $\mathcal{X}_{\mathcal{U}} \subset \mathcal{X}$ at each node l with step sizes $\eta^{(t)}$ by performing local state updates*

$$X_{\mathcal{U}}^{(l,t+1)} = \sum_{k=1}^{n} [W]_{lk} X_{\mathcal{U}}^{(k,t)} - \eta^{(t)} G^{(l,t)}, \tag{4.8}$$

where the gradient evaluated at $X_{\mathcal{U}}^{(l,t)}$ is

$$G^{(l,t)} = 2 \overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)^T} \left(\overline{\mathbb{L}}_{:\mathcal{U}}^{(l,t)} X_{\mathcal{U}}^{(l,t)} + \overline{\mathbb{L}}_{:\mathcal{A}}^{(l,t)} X_{\mathcal{A}} \right) \tag{4.9}$$

and W is defined in (4.7).

Proof. The online mirror descent technique in [47] is developed to track the minimizer of a global objective function (at each time t), where the global function can be written as a sum of m local functions. It is evident that the problem formulation given in Lemma 1 conforms to the setup proposed in [47]. Moreover, all the necessary assumptions of the method are satisfied when using the Euclidean norm as the Bregman Divergence.

Consequently, according to this method, the online distributed algorithm becomes

$$\begin{aligned} x_{u_l}(t+1) &= \arg\min \left\{ \eta_t \langle x, \nabla f_{l,t}(x_{u_l}(t)) \rangle + \frac{1}{2}||x - y_l(t)||_2^2 \right\} \\ y_l(t) &= \sum_{k=1}^{m} [W]_{lk} x_{u_k}(t+1), \end{aligned} \tag{4.10}$$

where x_{u_l} and $\nabla f_{l,t}$ are derived by vectorizing X_u^l and G^l in (4.8) and (4.9), respectively. Solving the above leads to (4.8), thereby concluding the proof. $\square$

Theorem 6 states that our localization problem may be solved by computing equation (4.8) in each network node. Notice that so far we defined every element of equation (4.8) except for the step size value $\eta^{(t)}$. Unfortunately, we can not apply the Barzilai-Borwein method in [67] because there is no distributed form for it. We looked for other possibilities unsuccessfully and are currently using fixed step sizes in the order of 10^{-3} or less. This affects the convergence rate of our algorithm and may be a point of further research.

4.4. Algorithm complexity

In order to compute the algorithm complexity of the noisy distributed case, as we assume $O(1)$ for each basic arithmetic operation, the gross computational complexity becomes the construction of matrix $\mathbb{L}^{(l,t)}$ at each time iteration. Therefore, we can assume the total complexity as $O\left(d^4 n \begin{pmatrix} n \\ d+1 \end{pmatrix} T\right)$, where T is the total number of iterations. Notice that we utilized the previous results of the centralized case, Section 3.4, in order to estimate this algorithm complexity.

Regarding the data exchanging complexity of the distributed case, we reiterate that each matrix $\mathbb{L}^{(l,t)}$ can be written with only one hop communication, *i.e.* each node only needs information that it can gather by itself and information that its neighbors have accumulated. Therefore, by only exchanging information among direct neighbors, each node is able to collect information of its neighbor's neighbors.

CHAPTER 5 : SIMULATIONS

In order to provide examples, visualize and compare solutions we sought the utilization of simulations, which we present and discuss in this chapter. All simulations were coded using Mathworks' Matlab environment. The main focus here is to compare our proposed algorithms, centralized GBC - Generalized Barycentric Coordinate and centralized/distributed GBM - Generalized Barycentric Matrix algorithms, with an existing MDS - MultiDimensional Scaling algorithm. As was explained in Chapter 3, our implementation of the centralized GBC follows the approach proposed by Diao *et al.* in [33], except we utilize Cayley-Menger bi-determinants to compute barycentric coordinates.

While we coded our own algorithms using basic Matlab functions resulting in a possible sub-optimal implementation, we utilized Matlab's MDS implementation, which should be more optimized. We also implemented the standard version of DILOC [32], which may also be sub-optimal in terms of run time and algorithm complexity as well.

The following sections will first provide simulation results for centralized GBC and GBM algorithms, comparing the first with DILOC and Matlab's MDS; and the second with Matlab's MDS. Then, later we provide simualtions for our distributed GBM algorithm showing RMSE values over time for multiple localized networks. We discontinued our comparison with DILOC as its location estimates are proven to converge to the ground truth in [32] for nodes inside the convex-hull of the $d + 1$ anchors. Therefore, for most completely random networks simulated by us, the number of nodes which satisfy this restriction is too little. Meanwhile, the MDS algorithm is able to localize nodes independently of their positions in relation to anchors, which is similar to our proposed algorithms, making it a better candidate for comparisons.

5.1. Noiseless centralized algorithms

As mentioned, all centralized algorithms are compared with Matlab's MDS function called *mdscale(.)*. We set its parameters as follows: its start condition is random, maximum iterations is set to one thousand and the stress criterion is squared and normalized with the sum of 4^{th} powers of dissimilarities. It is also possible to choose the number of replications, *i.e.* the number of random re-initializations it is allowed to do in order to obtain solutions with least stress. As will be shown by selecting the number of replications to one, each random initialization of MDS may provide a distinct localization estimate. These distinct solutions will have different stress values. Zero stress means perfect solution, with higher stresses meaning worse estimates. So, one usually runs MDS multiple times in order to choose the least stress estimate as the final solution.

In contrast, our centralized algorithms will either provide a completely correct solution or they won't. It will depend on the network structure. We emphasize again that each unlocalized node must have at least $d + 1$ neighbors which form a complete subgraph by themselves.

5.1.1. Generalized barycentric coordinates

For the following comparisons, we chose to construct a random network formed from a $6 \times 6 \times 6$ unit-spaced regular lattice in which independent Gaussian noise with zero mean and unit variance was added to each node coordinate. The maximum range threshold for each node i was set up to $r_i = 3$ units. Instead of a network graph for this example, we provide a histogram of node degrees in Fig. 7. Lastly, anchor nodes were manually chosen such that there were some nodes inside and outside their convex-hull.

From the 216 nodes in our example, 215 were correctly localized by our GBC algorithm, as shown in Fig. 7. Black circles and squares represent their true node coordinates for unlocalized nodes and anchors respectively; while red dots represent computed estimates. The unlocalized node has its true coordinates represented by the sole black star.

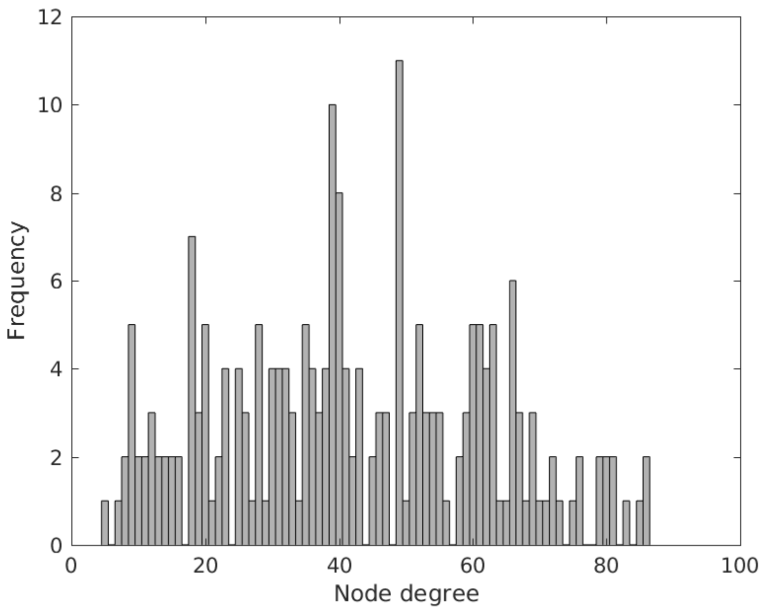
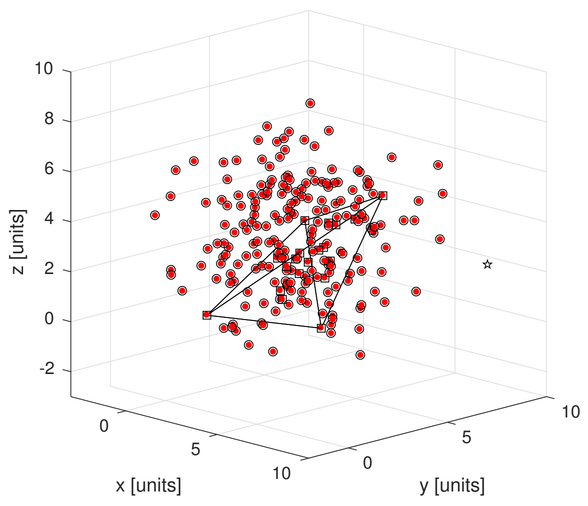


Figure 7: GBC algorithm's network example: node degree histogram.

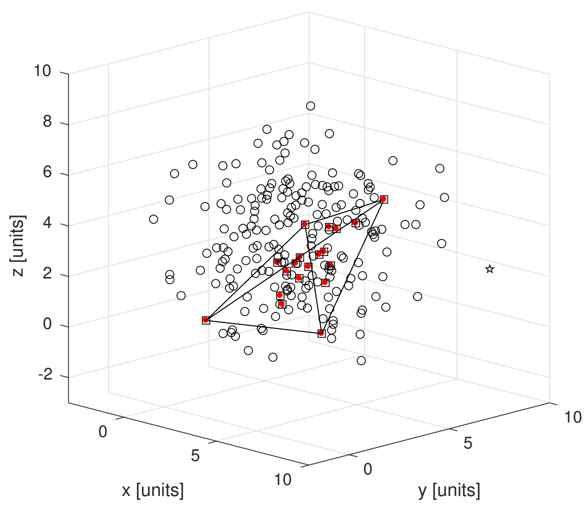
As one can expect, DILOC's standard algorithm is able to localize all nodes inside the convex-hull of the anchor nodes as shown in Fig. 8b. Though, in order to accomplish that, 6 of the 15 unknown nodes inside the convex-hull had to use measurements greater than our proposed maximum range of 3 units. If one limits DILOC's range to 3 units, some nodes would not be localized, as these nodes would not have the necessary number of neighbors to compute their barycentric coordinates. This result completely satisfies the specifications and assumptions of DILOC's proposed utilization and theory.

On the other hand, we simulated Matlab's MDS's algorithm 1000 times with one replication each time. A histogram of MDS's stress values is shown in Fig. 9a. This histogram was automatically generated using the square root binning method. It demonstrates the probability distribution of MDS stress value for this network, one can also skip to Figs. 13a and 13b and analyze the differences between this histogram and those of different networks.

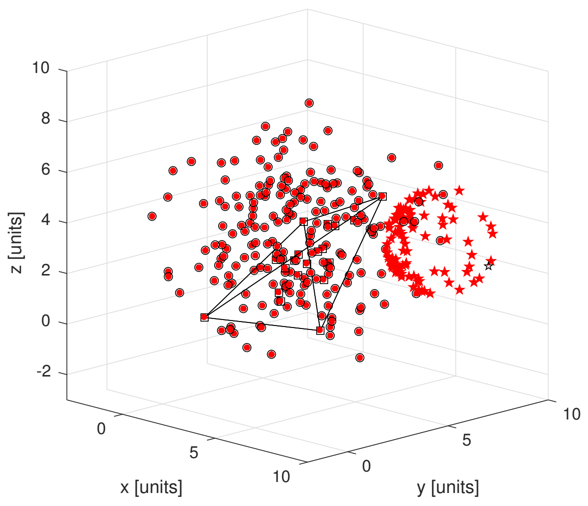
Fig. 8c shows all MDS computed coordinates which have a stress value inside the lowest



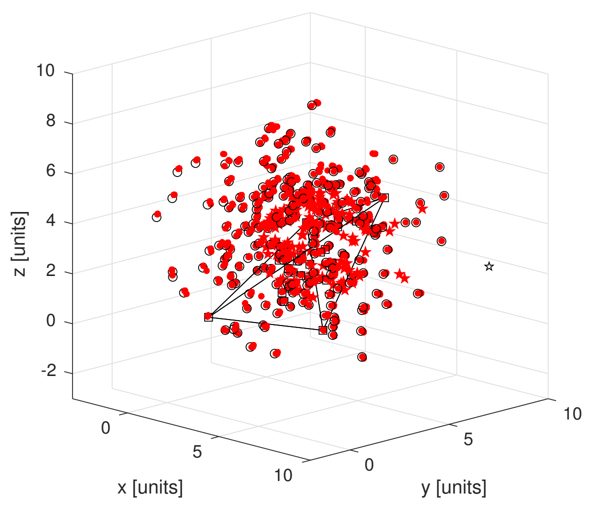
(a) Our generalized barycentric coordinate algorithm's location estimates .



(b) DILOC's location estimates.

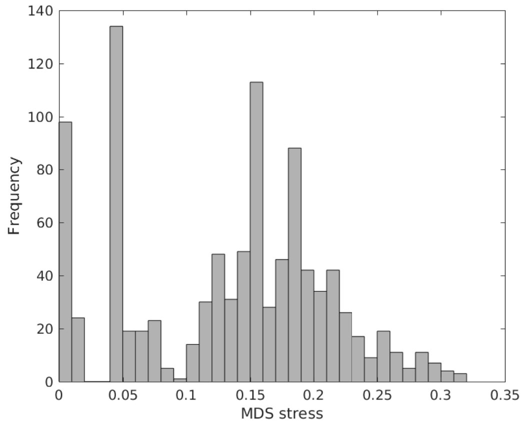


(c) Matlab's MDS's location estimates with least stress.

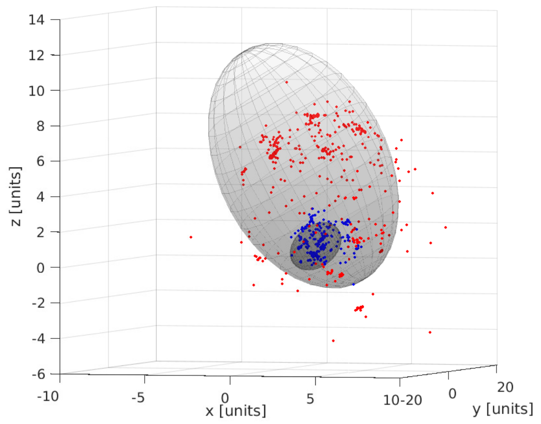


(d) Matlab's MDS's location estimates with most common stress interval.

Figure 8: Location estimates comparison: GBC, DILOC and Matlab's MDS. Black empty circles, squares and star are ground truth locations for each node. Black lines define the convex-hull of anchors. Red filled circles and stars are location estimates. Circles used for localized nodes; stars for unlocalized nodes.



(a) Matlab's MDS stress value histogram for 1000 simulations with one replication each.



(b) Smallest and largest covariance matrices error ellipsoids for Matlab's MDS's estimates.

Figure 9: Some characteristics of Matlab's MDS algorithm.

valued bin interval from the histogram shown in Fig. 9a. In this case, except for one node with computed coordinates marked by red stars, all others were correctly localized. Notice that, each red star represent one possible solution in which the MDS stress criterion was less than 0.01, which shows that MDS solutions are not necessarily unique. Moreover, according to the stress value histogram, this best case scenario happened in less than 10% of all simulations. Lastly, the computed node coordinates for the most frequent stress value interval are shown in Fig. 8d.

A visual representation of the best and worst computed node coordinates throughout all 1000 simulations of Matlab's MDS can be seen in Fig. 9b. We used [68] to obtain error ellipsoids which are centered at the average node location estimations. These ellipsoids have axes and orientation proportional to the covariance matrix eigenvalues magnitude and eigenvectors orientation. Their volumes are computed so that the confidence interval is 90%. Such results demonstrate that this algorithm may provide less than desirable results, moreover there is a large difference between optimal and sub-optimal estimations. Therefore, it is important to utilize a considerable amount of replications.

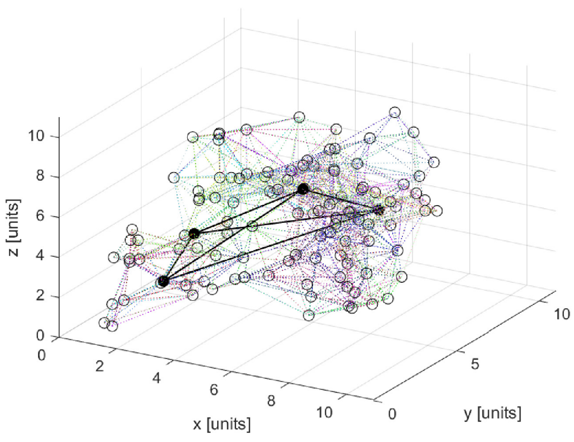
In order to provide an idea of our algorithm's run time and compare it to Matlab's MDS implementation, we executed 100 simulations of the former and 1000 for the later. Each MDS simulation was done with one replication only. The average run time for the first part of our GBC algorithm which constructs possible families of neighbor sets is 140.13 seconds with a standard deviation of 2.71 seconds. The second part which constructs and solves the linear system takes in average 49.15 seconds with a standard deviation of 1.32 seconds. The average execution time for one run of Matlab's MDS implementation with only one replication was 2.80 seconds with standard deviation of 0.71 seconds. Although our run times are 68 times larger than MDS' single replication rounds, one will usually need to compute a few rounds of MDS in order to obtain sufficiently small stresses, as can be inferred from Fig. 9a. This behavior states that our algorithm run times are not as bad as mentioned, which is the worst case scenario for our comparison one in which MDS solves the system in one iteration.

It is interesting to notice that one node, represented by a black star in Fig. 8, was not localized by any method. Besides being outside the anchor's convex-hull, it has less than $d+1$ neighbors given the defined maximum measurement range. Therefore, it does not satisfies either DILOC's or our algorithms assumptions and restrictions. According to our beliefs, it is better to provide a negative answer, *i.e.* it can not be localized, than to provide a wrong location.

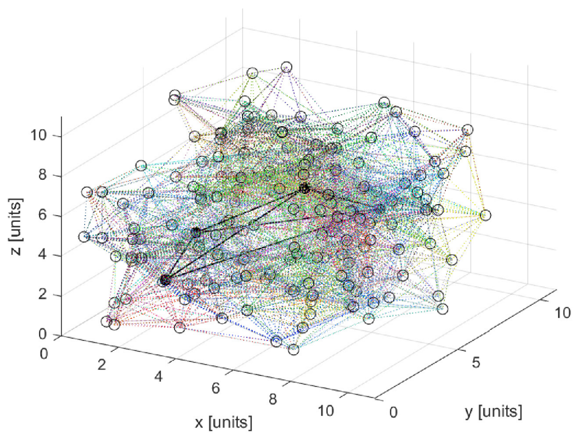
Next, we compare our proposed GBM centralized algorithm w and Matlab's MDS.

5.1.2. Generalized barycentric matrix

Now, for the following simulations, we chose a different method to create random networks. First, we chose 175 random points in three dimensions. Then, after selecting the measurement and communication range radius, $r_i \, \forall i$, of 3 and 4 units respectively, we test the constructed networks for constraints like node degree and iteratively exclude nodes that are not localizable. The resulting networks are shown respectively in Figs 10a and 10b, the former with 115 nodes and the later with 174 nodes. We emphasize that by construction the second network contains the first. Therefore, we chose to use the same randomly picked anchor nodes for both.



(a) Radius of 3 units.

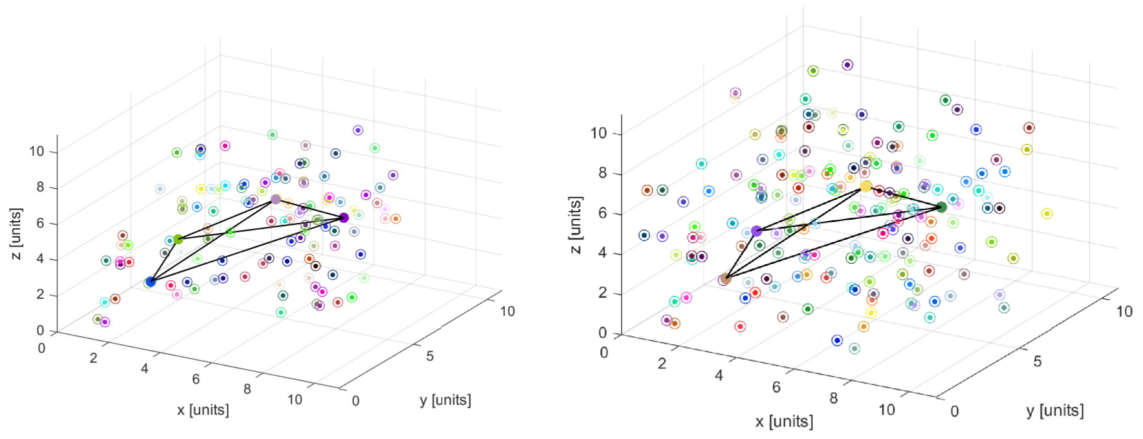


(b) Radius of 4 units.

Figure 10: Centralized GBM algorithm's network examples: network graphs. Anchor locations and convex-hull marked by black filled circles and black continuous lines. Black empty circles mark ground truth locations. Dashed lines represent network range sensing and communication connections.

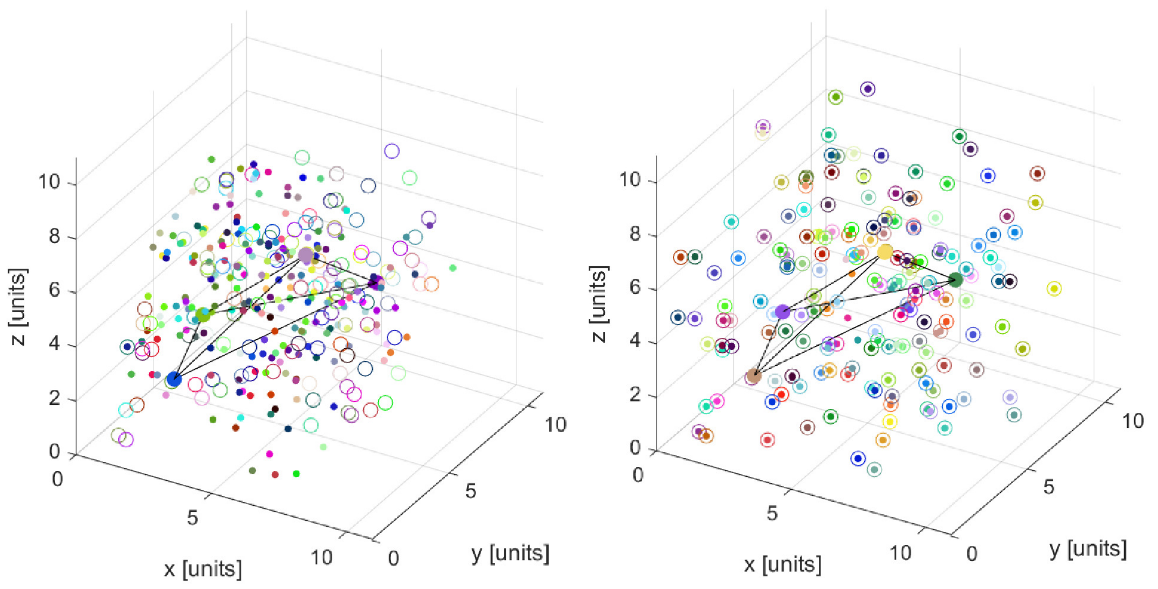
In Figs. 11 and 12, we provide simulations of our proposed GBM algorithm and Matlab's MDS implementation, $mdscale(.)$, respectively for the networks in Fig. 10. It can be seen in Figs 11a and 11b that our noiseless centralized algorithm is able to localize all nodes of both networks correctly. On the other hand, Matlab's MDS estimation for the first network with range radius of 3 units is sub-optimal, as seen in Fig. 12a. While, its solution for the second network is optimal, Fig. 12b. In these figures, each large empty circle of some color is the ground truth location of the respective node, while the small filled circle of the same color is the estimated solution provided by the respective algorithm.

Different from the simulations presented before in section 5.1.1 which uses 1000 replications, here we run only 100 resulting in the stress histograms shown in Fig. 13. Similar from the previous simulated network, they show that a majority of solutions provided by the MDS algorithm have non-zero stress, *i.e.* solutions have significant deviation from ground truth. But, differently from before, the actual number of estimated solutions with zero stress



(a) Radius of 3 units. (b) Radius of 4 units.

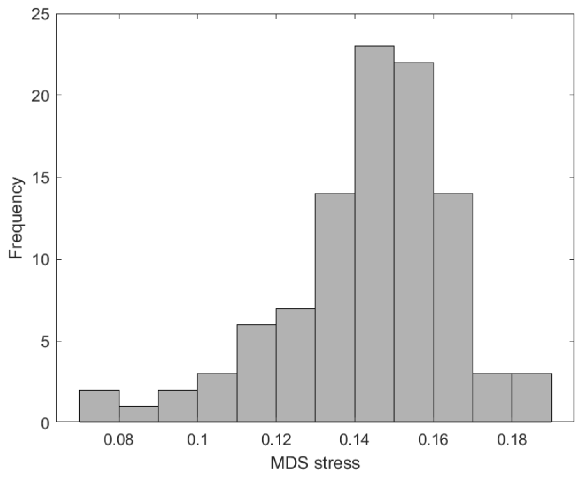
Figure 11: Centralized GBM algorithm's solution estimates. Anchor locations and convex-hull marked by black filled circles and black continuous lines. Colored empty and filled circles mark ground truth locations and location estimates; where each node has its own distinct color.



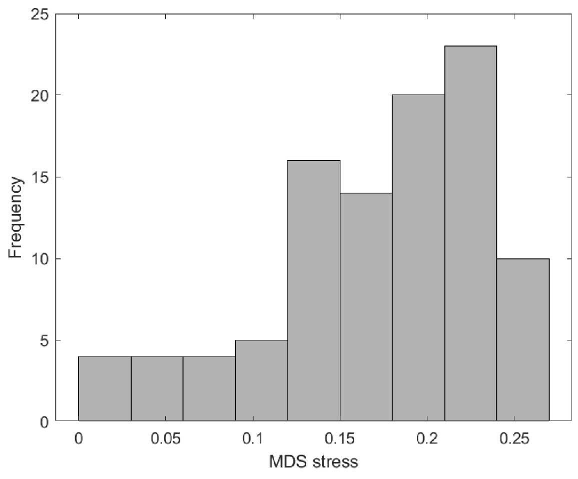
(a) Radius of 3 units. (b) Radius of 4 units.

Figure 12: Matlab MDS's solution estimates. Anchor locations and convex-hull marked by black filled circles and black continuous lines. Colored empty and filled circles mark ground truth locations and location estimates; where each node has its own distinct color.

is either zero, for the first network or less than 5% for the second, while the previous case had around 10% of correct solutions.

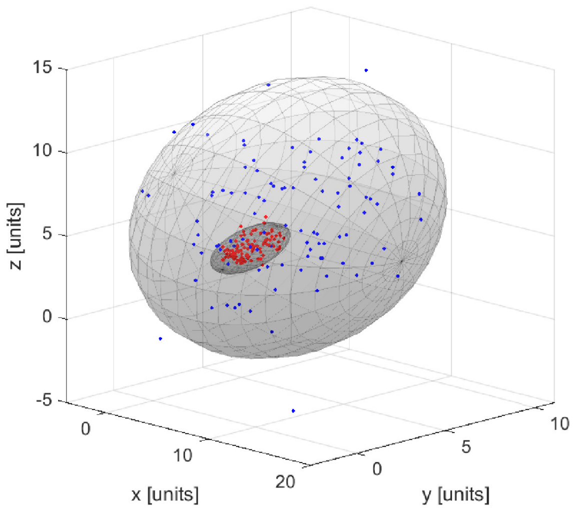


(a) Radius of 3 units.

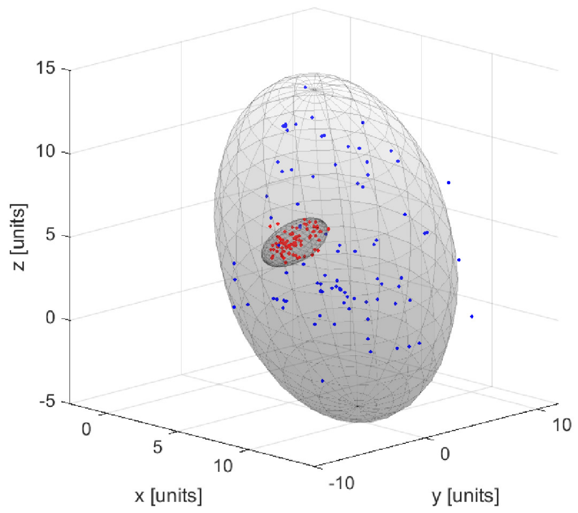


(b) Radius of 4 units.

Figure 13: Matlab's MDS stress value histogram for 100 simulations with one replication each.



(a) Radius of 3 units.

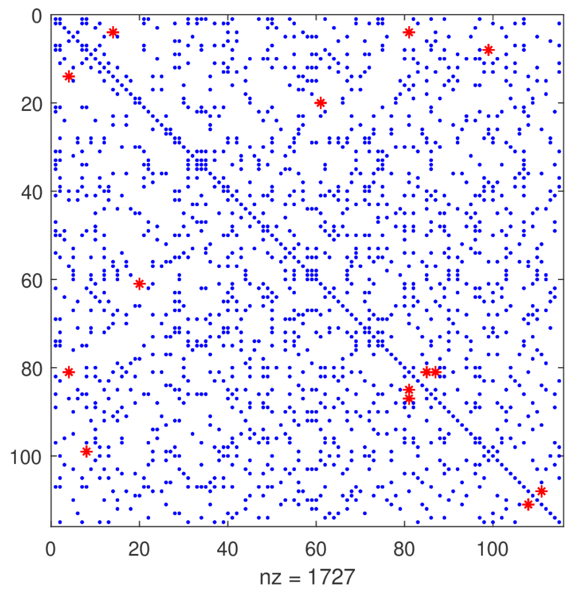


(b) Radius of 4 units.

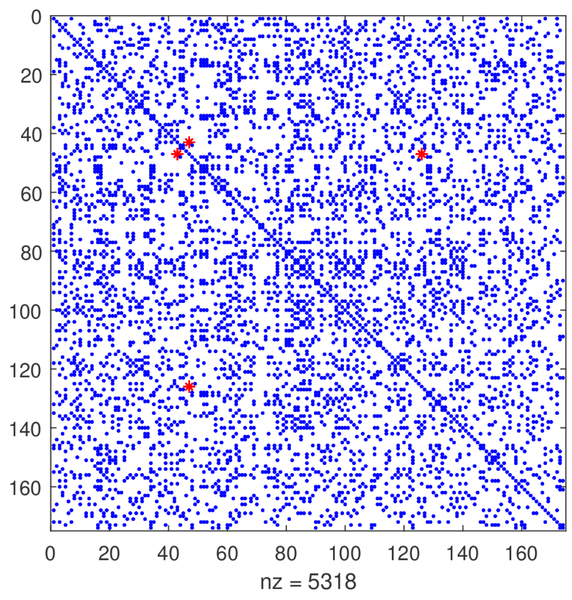
Figure 14: Smallest and largest covariance matrices error ellipsoids for Matlab's MDS's estimates.

Again, in order to provide a visual representation of the best and worst computed node coordinates throughout all 100 simulations of Matlab's MDS, Figs. 14a and 14b, we use again [68] to obtain error ellipsoids. As can be seen, we obtain similar results as the previous case.

During our tests, each iteration of the MDS algorithm run in average for 2.53s with standard deviation of 0.89s for the smaller network and an average of 3.15s with standard deviation of 0.85s for the larger one. In comparison, our algorithm took 1.4s and 43.96s for each network respectively. As we utilize all possible combinations of node neighbors, the run time of our algorithm will increase with the number of nodes and the number of neighbors of each node. Our GBM algorithm runs faster than our GBC algorithm as we utilize Matlab's implementation of n choose k combination function for GBM and not for GBC. This happened because we tried to create a function which selected a determined number of nodes based on depth or breadth first algorithms, which we manually implemented using *for* loops. Therefore, our GBC implementation is less optimized than our GBM one.



(a) Radius of 3 units.



(b) Radius of 4 units.

Figure 15: Non-zero elements of matrix W, communication weight matrix. Blue dots are elements which it has in common with the respective $\sum_{l=1}^{n} \mathbb{L}^{(l)^T}\mathbb{L}^{(l)}$ matrix, while red asterisks are distinct.

Finally, Figs. 15a and 15b show the non-zero elements of weighted communication matrices W, equation (4.7), for their respective networks. The blue dots are elements it has in common with its $\sum_{l=1}^{n} \mathbb{L}^{(l)^T} \mathbb{L}^{(l)}$ matrix, while red asterisks are distinct. The first network has matrix W with 1727 non-zero elements of 13,225 total elements, from which 14 are not part of its $\sum_{l=1}^{n} \mathbb{L}^{(l)^T} \mathbb{L}^{(l)}$ matrix. For the second network, we have respectively 5318 non-zero elements of 30,276 total elements, with 4 distinct ones. These different elements are related to barycentric coordinates with zero value.

5.2. Noisy distributed algorithm

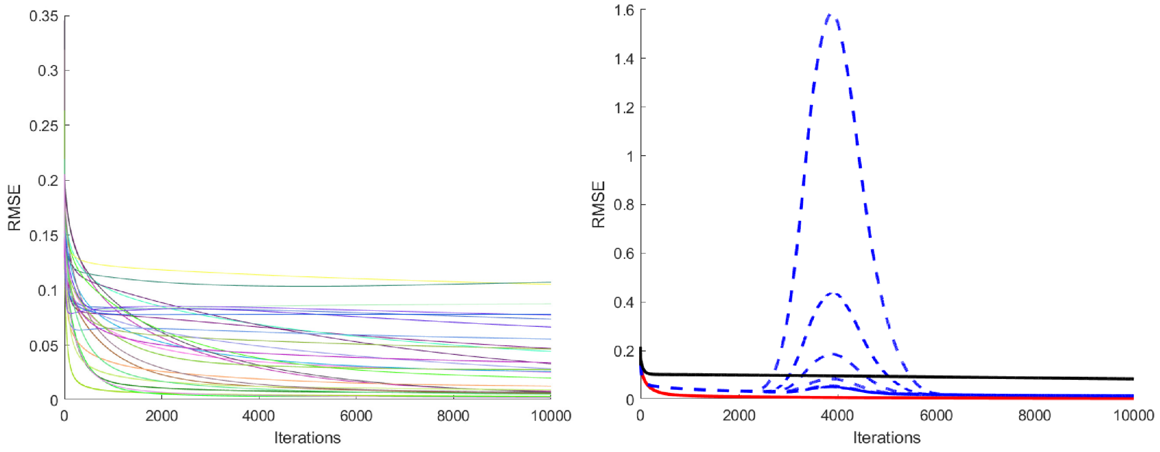
In this section, we present simulated results for our proposed distributed GBM algorithm. In order to do so, we generate 32 random networks with 14 nodes, *i.e.* 4 randomly chosen anchors and 10 unknown nodes, similarly to what was done in Section 5.1.1. All these nodes are distributed in a cube with side equal to 10 units. As we wanted to ensure that every unknown node would be localized, we chose a range radius of 9 units for all networks. All simulations were made with constant step size $\eta^{(t)} = 1.0 \times 10^{-3}$, as there is no apparent way to use dynamic step sizes with our distributed algorithms.

The added noise to each range measurement has Gaussian distribution with zero mean and a tenth of the true distance between respective nodes for its standard deviation. This approach means that farther nodes will have worse range measurements, which is a logical assumption. We show in Section 5.3 via simulation and RMSE - Root Mean Square Error values that choosing constant or proportional variance will not influence the behavior of our distributed GBM algorithm.

Fig. 16 shows the RMSE for each network node location estimates. From those 32 networks, we selected networks 1, 20 and 21 to show different possible behaviors in Fig. 16b. In this figure, the red continuous curve represents the RMSE values over iterations for a network which converged to the desired solution (network 1), while the black continuous curve represents one that did not converge within the number of simulated iterations (network 21). This does not mean that our algorithm solution estimates for this network will never converge, it will probably just take more iterations. Meanwhile, the blue dashed curve shows what happens when oscillation occurs (network 20).

In order to better illustrate our solutions for these 3 selected networks, we show in Figs. 17 the solution over time, which we call trajectories, for each one of these networks. The network which arrived to its correct solution (network 1), the one that did not (network 21) and the one with oscillatory behavior (network 20) are drawn in Figs. 17a, 17b, 17c and 17d. We specially draw two Figs. 17c and 17d in order to show that oscillations will generate different trajectories for each node with large oscillation amplitudes near the node which is the source of this behavior.

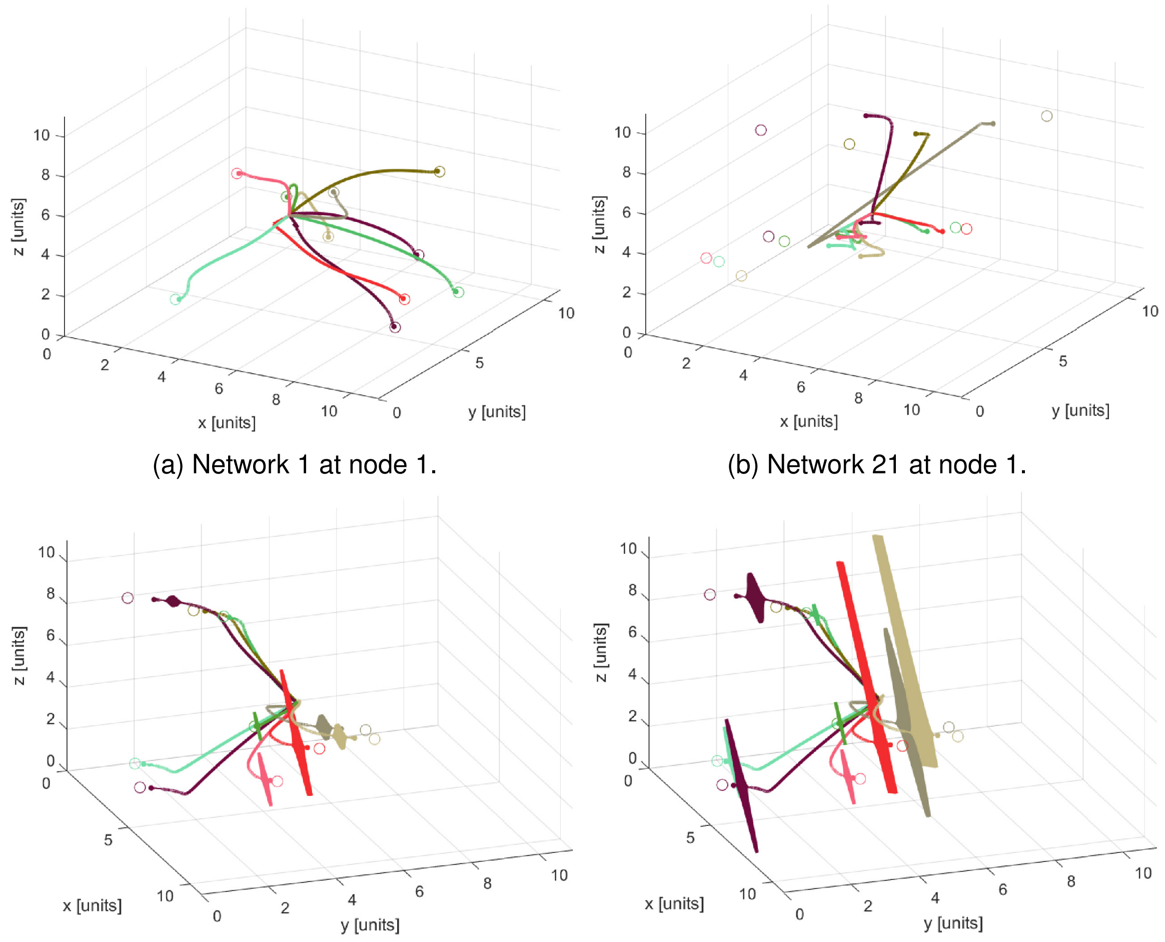
All simulated results for our proposed algorithms, be it centralized or distributed, are within expected. We emphasize that the distributed GBM algorithm will iterate towards the desired solution for as long as we allow it, despite possible oscillations that may occur. In order to speed up our distributed algorithm, one may try to apply some modifications to it like a variable optimal step size $\eta^{(t)}$ and reduction on the number of neighbor groups chosen for each node.



(a) RMSE values for 29 of 32 networks. Different colors indicate different networks.

(b) RMSE values for networks 1 (red continuous line), 20 (blue dashed line) and 21 (black continuous line).

Figure 16: Distributed GBM algorithm's RMSE estimates values over iterations.



(a) Network 1 at node 1.

(b) Network 21 at node 1.

(c) Network 20 at node 10.

(d) Network 20 at node 14.

Figure 17: Distributed GBM algorithm's solution trajectories over iterations. Colored empty circles mark ground truth locations for each node. Colored lines represent node estimates over iterations. Each node is represented by a different color.

5.3. Centralized versus distributed algorithms

We carried out simulations comparing the performance of the centralized two-phase averaging algorithm and the centralized gradient descent algorithm both explained in Sec. 4.2 respectively by equations (4.3) and (4.5) and the distributed online gradient descent in Sec. 4.3. Fig. 18 shows again a randomly generated static network with 4 anchors and 10 unknown nodes, whose ground truth position lie on a cube with side length of 10 units. The maximum measurement range of each node was $r_i = 8$ units and random Gaussian noise with variance equal to either a tenth of the actual distance between nodes or the averaged valued of these proportional variances were applied to the range measurements. Proportional variances characterize that larger distances have larger noise, while a constant one says that the sensors maintain the same amount of noise throughout its range. In this simulated case, our results had no significant discrepancies depending on these choices. The node coordinate estimates computed by the first two algorithms over time are also shown, as well as the estimates locally computed by the third algorithm in a randomly selected node.

There are two fundamental differences between the centralized and distributed algorithms. First, while the centralized algorithm has a single set of estimates for the unknown node locations, the distributed algorithm maintains a different set of locations estimates for each node in the network. Second, the amount of information available to each node in the distributed algorithm is different compared to the centralized case and depends on the network communication structure. These facts are clearly seen in the results. For example, note that the two-phase averaging and the centralized gradient descent algorithms provide estimate trajectories that remain close to each other, while some estimates obtained by the distributed gradient descent algorithm follow very different trajectories. Moreover, it is important to note that while the first two algorithms have the same initial estimate, the same does not occur for the distributed algorithm, despite computing initial estimates in similar ways. This behavior is due to the fact that the distributed algorithm uses only the available neighboring first noisy range measurements to initialize its estimates. Both centralized algorithms are able to provide better initial guesses for all node locations, while the distributed algorithm provides poorer results for nodes with fewer neighbors or at a further distance from the anchors.

Fig. 19 shows the Root Mean Square Error (RMSE) of the node coordinate estimates over time for the three algorithms. Both centralized algorithms arrive at satisfactory location estimates with less than 5000 iterations, while the same can not be said for the distributed

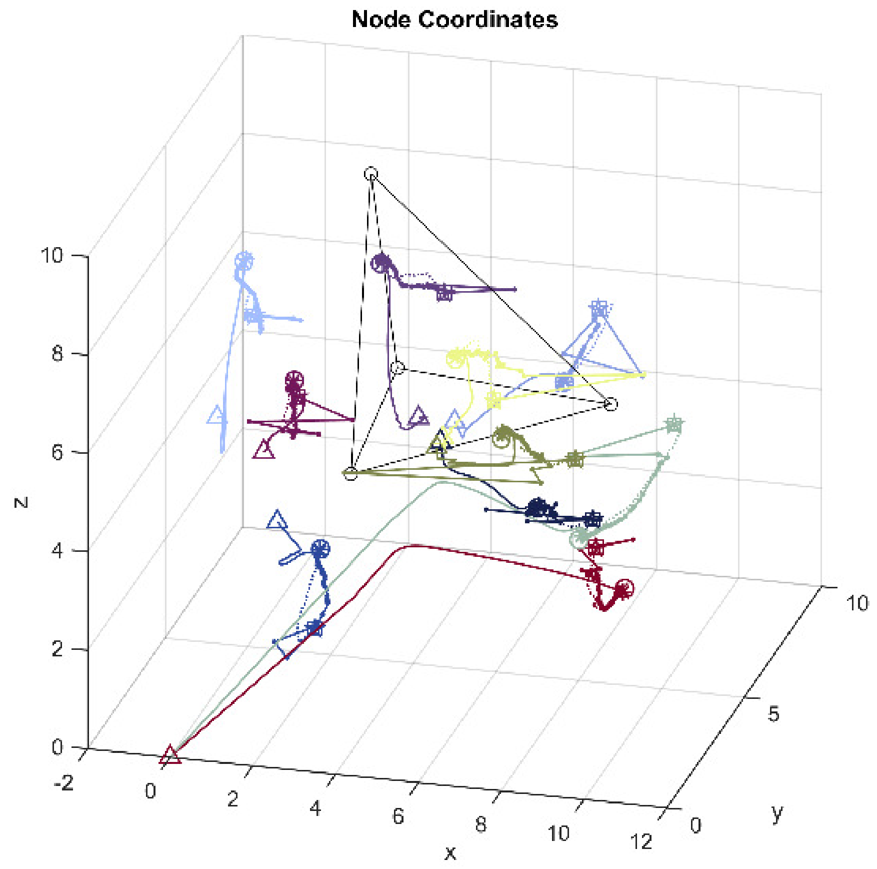
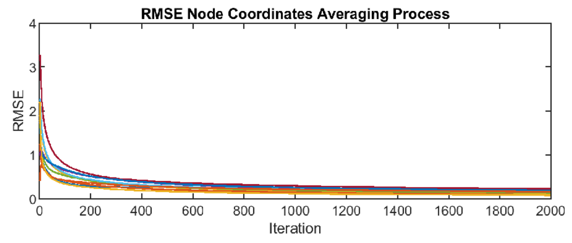
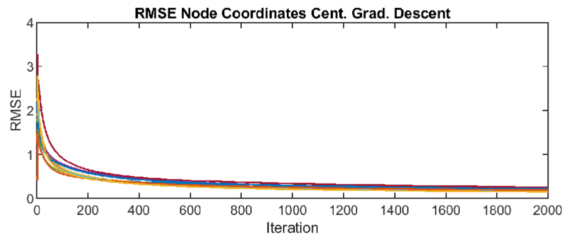


Figure 18: Randomly generated sensor network with 10 unknown nodes and 4 anchors and trajectories of the node coordinate estimates over time. Each color represents a different unknown node. The dotted, dot dashed and full lines beginning with $\{\square, \pentagon, \triangle\}$ and ending with $\{\diamond, \times, \ast\}$ show the estimates over time obtained by the two-phase averaging Sec. 4.1 in equation (4.3), centralized gradient descent Sec. 4.2 in equation (4.5) and distributed online gradient descent Sec. 4.3 algorithms, respectively. The ground truth node locations are marked by circles, while the black tetrahedron shows the convex hull of the anchor nodes. As each node in the distributed algorithm computes its own unknown node location estimates, it is not possible to show the trajectories of the estimates of all nodes. Instead, we show the trajectories of the estimated node locations computed by one randomly chosen node.

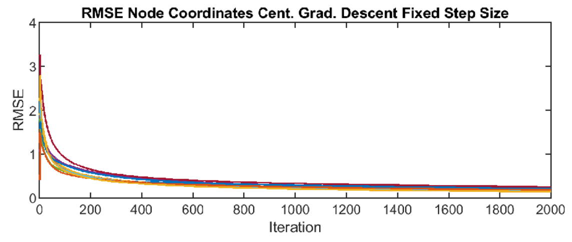
gradient descent algorithm. It is clear that the initial estimates in the distributed case are worse than the ones in the centralized algorithms, which contributes to its slower convergence. Another important factor that impacts our distributed gradient descent method is the choice of the learning rate, $\eta(t)$. A small learning rate will provide a more stable but slower convergence. So far, while we used an $\eta(t)$ based on the Barzilai-Borwein method [67] for



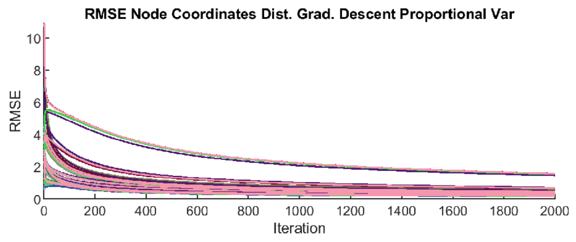
(a) Averaging process.



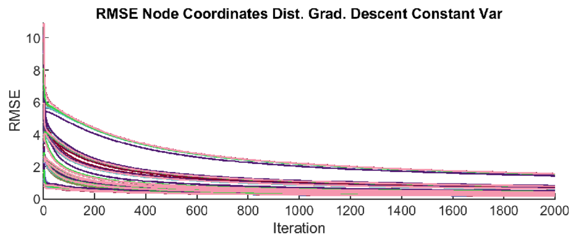
(b) Centralized gradient descent.



(c) Centralized gradient descent with fixed step size.



(d) Distributed gradient descent with proportional noise variance.



(e) Distributed gradient descent with constant noise variance.

Figure 19: Root Mean Square Error (RMSE) of the node coordinate estimates over time for the two-phase averaging Sec. 4.1 in equation (4.3), the centralized gradient descent with Barzilai-Borwein and fixed step sizes Sec. 4.2 in equation (4.5) and distributed online gradient descent with proportional and constant variances Sec. 4.3 algorithms. As each node in the distributed algorithm computes its own unknown node location estimates, we present RMSE values for the estimates of each node on the same plot.

the centralized gradient descent algorithm with good results, we were not able to compute $\eta(t)$ online in a way that provides faster convergence rates while guaranteeing stability of the process for different networks. Therefore, we used a small fixed η only for simulations involving our distributed online gradient descent algorithm.

CHAPTER 6 : EXPERIMENTS

We started our experiments in order to get a physical network of devices able to localize themselves by acting either as anchors or unknown nodes. The idea was to have a communication network mimicking our network model, where nodes directly share information with its neighbors. Moreover, devices' sensors should ideally be able to correctly and precisely range other devices, or at least, we should be able to create a model from which we could estimate correct range measurements as independently as possible from external influences, such as interference from other sensors and equipment, environment structure and materials and even weather variations. Finally, we desire to form a localization system which can be applied to other systems and devices, such as mobile robots. Off course, this last point would require our localization theory to enable mobile nodes, which is one of our possible research directions.

Despite not having complete experimental results because of sensor problems, we provide in this chapter the fundamental knowledge we obtained and developed. We start by specifying our requirements for both hardware and software in order for our localization problem to work. Then, we provide all chosen tools specifying some of their characteristics and how we applied them. Later, we provide our implemented algorithms in Appendix A.2.1.

Our main requirements for both software and hardware are:

- Communication network using mesh topology;
- Fast and simple communication setup and utilization;
- Range sensors should be able to measure its distance to other sensors, while being measured by another sensor, or have a fast sensing time and ability to switch roles online;
- Hardware able to run ROS - Robotic Operating System for better integration with other robotic environments.

There are three main hardware devices required for our purposes: small computer board with wireless network, range sensor and power supply. For the power supply we chose a generic rechargeable battery with standard USB connections, which allowed our system to be mobile for the required period of time. Next, we discuss board and sensor options.

6.1. Base board options and operational system

In order to use ROS - Robotic Operating System, one requires our choice of board to run newer versions of Ubuntu [69]. In our case we chose 14.04 LTS, as it was the newest compatible version for ROS at the time. For the newest compatibility specification one should consult ROS web page and its forums [70].

Nowadays, there are multiple possible small computer board systems, each with its own variations, capabilities and peripherals. But, at the time when we searched for an embedded development system with wireless communication, three systems were more common and had possibilities to satisfy parts of our requirements: BeagleBoards [71], ODROID [72] and RaspberryPi [73]. From these, we chose the Beagle Bone Green Wireless - BBGW [74], Fig. 20, as it sported the newest version of Texas Instruments WiFi module, which was theoretically able to connect to mesh networks. It seems to be one of the first entry level development boards to come with a WiFi module integrated in it. Its main specifications are CPU AM3358 1GHz ARM Cortex-A8, WiFi + Bluetooth module WLinkTM8 both from Texas Instruments, 512 MB of DDR3 RAM and 4GB of eMMC flash storage. It also has the most common connection interfaces such as I^2C, SPI, USART and USB.

This board already comes with a pre-installed Linux, but in our case, we found out when we first booted the board that its available WiFi driver was old and did not have mesh network capabilities. This fact made us search for ways to install a new driver which we tought would be simple enough, but it wasn't. In order to install the newest version of Texas Instruments driver at the time, we were required to download, compile and install the Linux kernel following instructions partially available at Texas Instruments website. We

Figure 20: Beagle Bone Green Wireless compute board. It sports a mesh enabled WiFi module.

say partially available because they only explain how to set up the driver package while one is configuring the new version of Linux to be compiled and after compilation. So we had to search for instructions on how to compile a Linux distribution for our ARM based hardware platform. Mind you that it is required to have files specific for the board being utilized. All these procedures are specified in Appendix A.2.1. As mentioned, they were developed based on Robert Nelson wiki [75] which explains how to compile Linux for various different development boards and Texas Instruments reference materials [76–79] which show us how to modify pre and pos compilation parameters so that their WiFi module will work with the newest driver.

We emphasize that once this procedure is done, on may simply re-install the compiled version of Linux to as many Beagle Bone Green Wireless boards as required. In order to do so, one must load the compiled kernel in a micro SD card and follow the different boot procedures specified at BBGW manuals [71], so that the new kernel will be copied to the board internal eMMC flash memory. Following which, one may either connect the board via USB to a computer and access its kernel terminal via SSH, or one can use an external monitor, keyboard and mouse directly connected to the board to proceed with the installation of ROS and any other necessary software via the standard apt-get functionality of Linux.

Lastly, one needs to configure all boards to work and communicate to one another using WiFi 802.11s the mesh standard for WiFi networks as see Appendix A.2.2. This configuration entails setting up each individual board with its hostname, its collection of ip addresses (USB virtual Ethernet port and WiFi), its mac addresses, a file with all of this information from the entire network (acts as a lookup routing table) and a initialization script to carry over all necessary procedures during booting process. This initialization entails bringing up all network ports and attributing the designated ip addresses to each board, so these files as individually created up to a certain point.

Next, we show our work around the two different sensors that we used unsuccessfully and explain what were the problems we had while utilizing them.

6.2. Range sensors

We started our sensor development and tests using Kempke *et al.* Polypoint range sensor [80] shown in Fig. 21. It is based on DecaWave radio that provides accurate time stamps for TOA - Time of Arrival and TDOA - Time Difference of Arrival based algorithms. In a high level view of the technology, one may explain TOA algorithms as based on the time of flight necessary for the electromagnetic wave to propagate from one sensor to another, which is later converted to distance by utilization of the waves velocity in a specific medium. TDOA works differently and requires one sensor to capture the time difference between two or more waves arriving at its sensing element, then with the previous knowledge of the locations of the origin points of all received waves, it is able to compute its distance in relation to them. Both technologies are affected by the existence or not of direct line of sight and wave reflections which may lead to incorrect measurements. In order to compensate for this, Kempke *et al.* compute an average of multiple measurements in the Polypoint before providing a range measure. While this procedure allows them to provide better estimates, it makes their overal system slower. Which may not be a problem for other

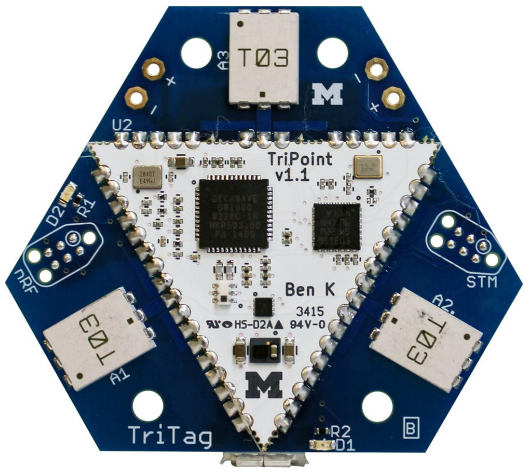


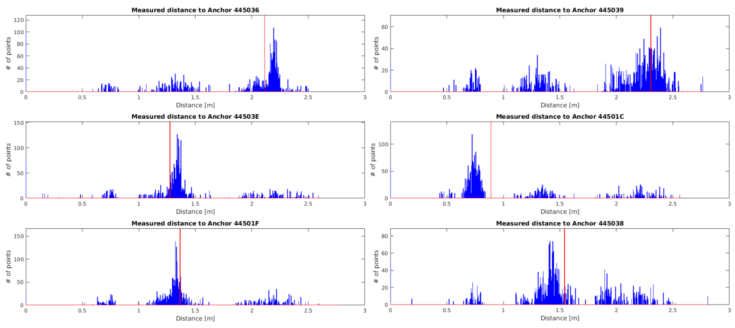
Figure 21: Polypoint Tritag range sensor board. It may act as an anchor or an unknown by switching its start-up sequence. In any case, the resulting ranging topology is a star with the ranging node in the center and the anchor in the extremities.

platforms, but is a nuissance for our algorithm as we also average all range measurements received.

We tested it inside a small enclosed lab space, with and without line of sight, Fig. 22 and 23 respectively, as can be seen in 23 the sensors may or may not communicate among themselves when there are objects or structures occluding their line of sight. Moreover, while such a problem did not occur in all of our line of sight experiments, we notice that sensors may provide wrong measurements once in a while as seen in the histograms in Fig. 22b where the red data is the ground truth range distance obtained from the installed Vicon system and the blue data is obtained from the Polypoints. Notice that for this network configuration, one may approximate the obtained distribution using four distinct Gaussian distributions. This result makes us believe that there is either interference from wave reflections or from other sensors in the network.

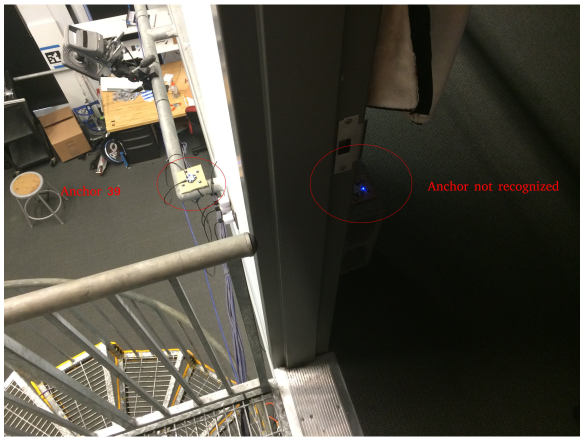
While the previous problem is a nuisance, it can be dealt with by computing these Gaussian distributions anch choosing the center of the one with highest peak value. On the other hand, these sensors worked in a star topology, *i.e.* one sensor would range itself in relation to all others, so while one sensor was ranging itself the others acted as anchors only providing reference points. In order to circumvent that, we tried to switch which sensor would range at a given time, but the procedure required a hardware reset and different start-up sequences each time we needed to switch from ranging to being ranged. This behavior was extremely detrimental for our work as we require a mesh topology where sensor should range themselves in brief periods of time so that one has the complete set of range measurements to run our algorithm.

(a) Image showing anchors and ranging device.

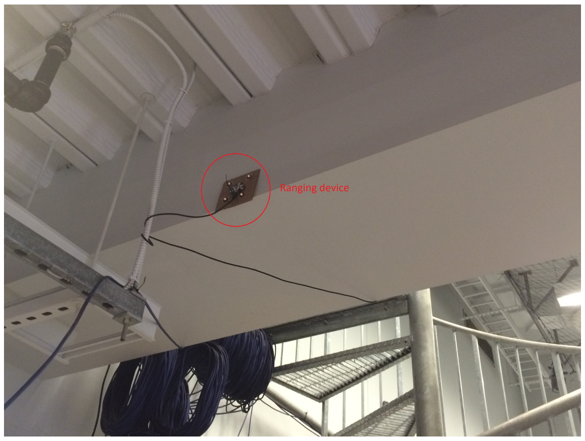


(b) Histogram plots showing the range measurements obtained from each anchor. Blue stacks show the quantities of range measurements per distance sets, while red lines indicate the ground truth.

Figure 22: Polypoint experiment arrangement were one Polypoint ranges itself in relation to all others which are acting as anchors.



(a) Image showing both anchors.



(b) Image showing device wich is ranging itself in relation to the anchors.

Figure 23: Polypoint experiment arrangement were only one anchor was recognized, while the other wasn't.

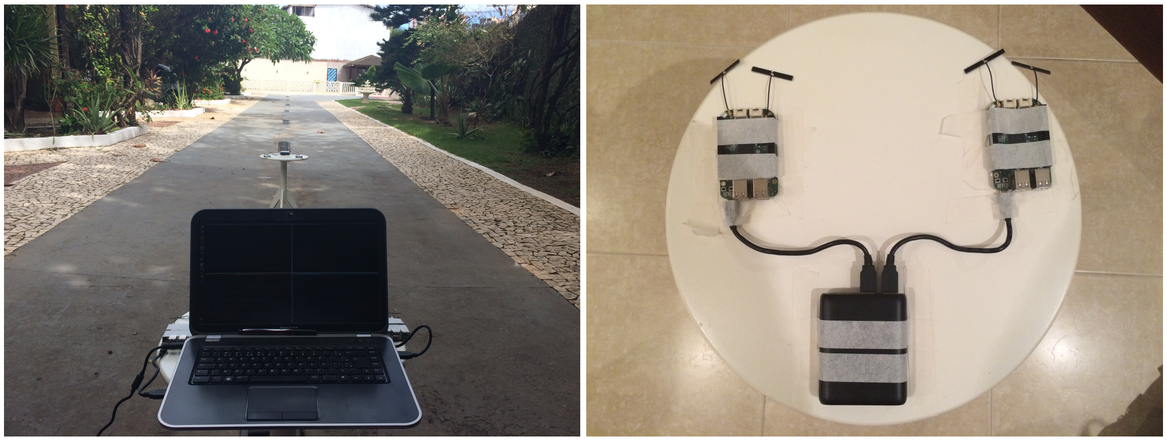
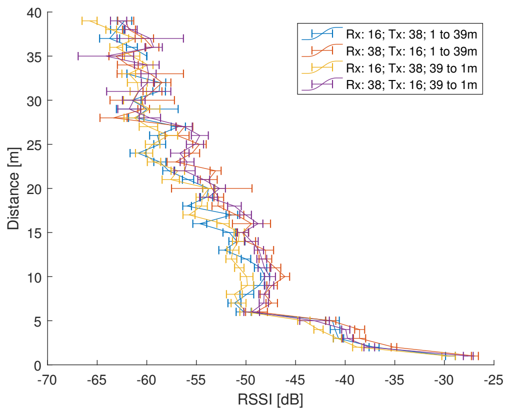

(a) Environment view of experiment. (b) Detail of mounting for moving boards.

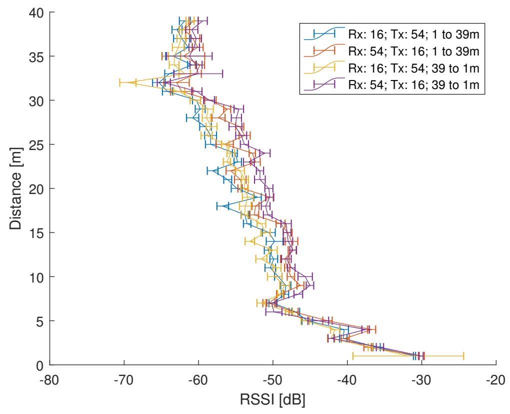
Figure 24: Capturing RSSI values every 1m from 1m to 39m in an open space with approximately 6m of width and 40m of length.

These problems motivated us to seek another sensor. After searching and studying existing sensors, we decided to start experimenting using the RSSI - Received Signal Strength Indicator obtained from Beagle Bone Green Wireless board WiFi module. We knew from literature that RSSI values also suffered from wave reflections and non line of sight problems, but there should be ways to circumvent those problems using sensor modeling in different environments. Therefore, in order to capture enough data to construct our models we searched for ways to constantly capture RSSI from each node of the network during a long period of time. From our analysis of the IEEE 802.11 protocol standard, we notice that each node of the wireless network would send beacon frames every 100 ms in order to inform other nodes of its presence in the network. Moreover, each of these frames is imbued with the RSSI values specific for each pair of transmitter and receiver, which is exactly what we were looking for. So instead of flooding our communication network with unnecessary data packets, we could use the already periodic beacon frame for our purposes.

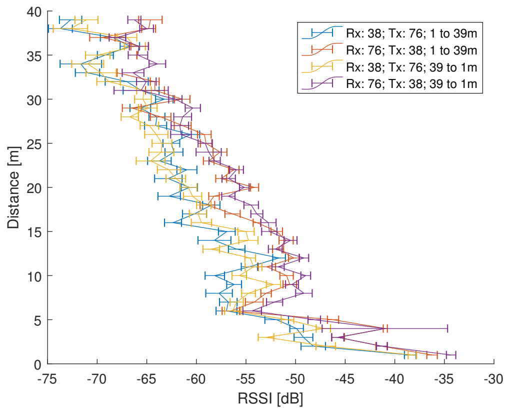
At first, we utilized the software package Wireshark in order to capture and analyze all received packets in a given WiFi module. This showed us that not all data packets carry RSSI values which reinforced our decision of utilizing only beacon frames in order to get these values. Moreover, we started using Wireshark to parse and store the relevant information which consisted of the transmitter and receiver pair ip and mac addresses and the respective RSSI value. But, Wireshark was too heavy of an application for our needs, so we explored their software libraries and found the freely distributed Radiotap library [81]. This library in written in standard C computer language and enabled us to create our own Ethernet packet capture and parsing software inside of the ROS environment, which was exactly what we needed.



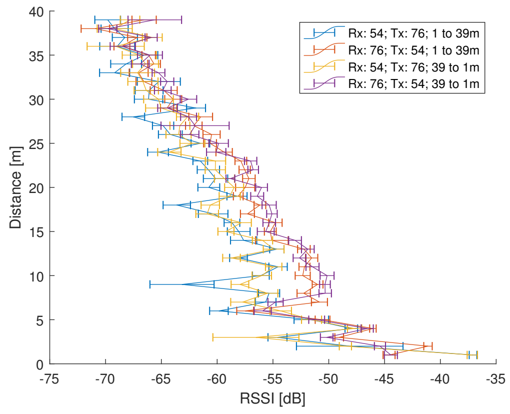
(a) Pair (16,38).



(b) Pair (16,54).



(c) Pair (38,76).



(d) Pair (54,76).

Figure 25: Experimental results from measuring RSSI values over distance. We show the inverse, Distance over RSSI, as this better represent our application, where we obtain an RSSI value and estimate a distance. Notice that in general the curve is well defined, but for each element there is large variation.

Using our developed data capture and the *rosbag* packages, we tested this setup under two main conditions: one was an approximately rectangular open space with 6 m of width and 40 m of length, the other was inside the previously used laboratory. In the former, we used four Beagle Bone Green Wireless boards, two fixed at a static base and connected to a computer via USB and two connected to mobile base. By moving the mobile base one meter at each iteration, we were able to gather the data shown in Fig. 25. If one looks at the plotted data with x-axis and y-axis respectively equal to inter-node distance and RSSI values, one may believe that the gathered data is actually quite good, as one can approximately fit it to create a model easily. But, one must pay attention that we actually want the inverse function, from RSSI values to range measurements, which is actually shown in Fig. 25.

At first and based on the results shown in Fig. 25, we decided to further develop our application towards this type of sensor. We had already tought about fitting RSSI to some model, which would possibly even required one fitting for each pair of devices, something not optimal, but not so different from the previous Polypoint which needed to be calibrated individually. But further tests in enclosed space, the same lab were we experimented on Polypoint, showed that RSSI values largely vary because of, for example, line of sight obstruction by different materials, board reset, board altitude (its distance in relation to the floor and the ceiling) and radio interference. These behaviors showed us that RSSI values may be better applied for open environments with few obstructions, which was not our main target as we wanted to develop a physical experimental set up for indoor localization.

After these experiments, we sought a new sensor but at the time, most ranging sensors had either the same problems as our tested devices, or worked under more severe limitations such as small distances and only point-to-point ranging capability. Further investigation on newer sensors or different technologies is needed for future research.

CHAPTER 7 : CONCLUSION

We started this work with the intention to provide a complete methodology and physical experimental setup to solve the problem of range-only localization in wireless sensor networks. We were able to realize a considerable portion of our intentions, which we presented in this dissertation. From a theoretical perspective, we presented a closed-form expression for computing barycentric coordinates in arbitrary d-dimensional node configurations, for which we utilized Cayley-Menger bi-determinants. These bi-determinants are a more uncommon form of Cayley-Menger determinants which is more widely used to solve localization problems via barycentric networks as previously explained. Our novel application of bi-determinants largely simplified the existing theory for computation of barycentric coordinates on arbitrary node arrangements, moreover we were also able to extend these results to the unconstrained d-dimensional case.

The previous proposed solution for barycentric coordinate computation allowed we to develop a centralized algorithm capable of handling noiseless range-measurements to solve our problem in, what can be called, a single step. From gathering noiseless range measurements, one compute all possible barycentric coordinates for each node, aggregate them in a over-determined linear system which we finally solve using a least-squares approach. In accordance to network topology, this method either provides the correct solution, or it is not able to do so. We view this strict binary behavior as a good approach, as one may not be misled by bad estimates; and in the case of a lack of solution, one may investigate its network to exclude unknown nodes that are unlocalizable by our theory, the ones that do not have sufficient amount of neighbors or are in a sub-graph of the network which may be segregated by cutting at most $d + 1$ edges.

From the existing theory, we identified possible methods to handle noisy range-measurements. Our first approach was to use averaging methods following the Law of Large Numbers from probability theory to obtain range averages over time, or iterations, that have less or even almost no noise after a certain amount of iterations occur. With these new averaged measurements, we could apply our centralized method at each iteration looking for its output which unfortunately presented with oscillating amount of noise, at one iteration it would have less noise then more noise and *vice versa*. We again applied an average process to subside this behavior resulting in better solutions over time for the centralized algorithm.

Our last theoretical and simulated result is related to the application of a distributed optimization algorithm to our proposed centralized method. Through its application, we developed a distributed gradient descent algorithm able to handle noisy range measurements and localize all unknown nodes in all nodes of the network with only local, 1-hop, communication and 2-hop information. A node compute its estimate and archive it and its ranging data, which it exchanges with its neighbors, so one node also has some information about a node 2-hops away.

We provided numerical results from simulated networks in both environments for both centralized and distributed algorithms. We also utilize this results to compare with other algorithms. Finally, we move on to our construction of the building blocks for future physical experiments. Unfortunately, given some sensor constraints such as large measurement variation over environmental changes, problems with non-line of sight, wrong ranging topology; we were not able to conclude our experiments. Despite these problems, we still provide solutions for a future attempt with new sensors which are the initial setup and configuration of chosen base boards from kernel download and compilation to wireless communication model driver configuration and IEEE 802.11 packets sniffing using ROS.

7.1. Future work directions

Future work should at first be focused on trade-offs between computational efficiency and localization accuracy by analyzing the effect of the number of neighbor subsets of size $d + 1$ used for barycentric coordinate computations. As these susbsets are composed via combination algorithms, larger networks and networks with high average node degree may not be localized by our algorithm in a timely manner. On the other hand, larger number of subsets will increase the probability of the correct localization of the network. Therefore, researching an algorithm capable of ensuring localizability while using the least amount of subsets is essential for the deployment of this algorithm for large networks.

As previously mentioned, mobile and static heterogeneous networks are becoming increasingly common. Meanwhile, our algorithm only provides a solution for static networks. While there are other solutions utilizing the constraint of nodes must be inside the convex-hull of the anchors, there are no current barycentric coordinate based algorithm for mobile networks with arbitrary anchor placement. This may be due to complications that arise from allowing the presence of values outside the range $[0, 1]$ in the barycentric coordinates (outside the convex-hull) which disallow the use of Markov Processes for its investigations.

Lastly, one may also investigate the possibility of utilizing mixed bearing and range measurements with barycentric coordinates. This approach may provide novel methods and theories as well as allow the utilization of mixed sensor types, enabling more measurements in the same amount of time, diminishing the noise present in the system.

APPENDIX

A.1. Cayley-Menger Bi-determinant proof

Following the approach given by [65], we begin by defining matrices $P = [\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_d]$ and $Q = [\mathbf{q}_0, \mathbf{q}_1, \cdots \mathbf{q}_d] \in \mathbb{R}^{d \times d+1}$, as

$$V_P = \begin{bmatrix} P \\ \mathbb{1}^T \end{bmatrix} \text{, and} \quad V_Q = \begin{bmatrix} Q \\ \mathbb{1}^T \end{bmatrix},$$

where $\mathbb{1}$ is the column vector of all ones with the appropriate dimension.

The signed volume of the sets of points $\mathcal{P}$ and $\mathcal{Q}$, specified by their coordinates given by the columns of matrices P and Q, can be found through the determinant of V_P and V_Q respectively:

$$\left| V_P \right| = (d!)\text{Vol}(\mathcal{P}) \tag{A.1}$$

$$\left| V_Q \right| = (d!)\text{Vol}(\mathcal{Q}) \tag{A.2}$$

Next, we can apply a sequence of operations without modifying the value of the determinants of V_P and V_Q.

$$\left| V_P \right| = \begin{vmatrix} P \\ \mathbb{1}^T \end{vmatrix} = \begin{vmatrix} P & \mathbf{0} \\ \mathbb{1}^T & 0 \\ \mathbf{0}^T & 1 \end{vmatrix} \tag{A.3}$$

$$\left| V_Q \right| = \begin{vmatrix} Q \\ \mathbb{1}^T \end{vmatrix} = \begin{vmatrix} Q & \mathbf{0} \\ \mathbb{1}^T & 0 \\ \mathbf{0}^T & 1 \end{vmatrix} \tag{A.4}$$

For the determinant of V_P, we take its transpose and interchange its last two columns and last two rows, obtaining:

$$\left|V_P^T\right| = \begin{vmatrix} [\mathbf{p}_0]_1 & [\mathbf{p}_0]_2 & \dots & [\mathbf{p}_0]_d & 0 & 1 \\ [\mathbf{p}_1]_1 & [\mathbf{p}_1]_2 & \dots & [\mathbf{p}_1]_d & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ [\mathbf{p}_{d-1}]_1 & [\mathbf{p}_{d-1}]_2 & \dots & [\mathbf{p}_{d-1}]_d & 0 & 1 \\ 0 & 0 & \dots & 0 & 1 & 0 \\ [\mathbf{p}_d]_1 & [\mathbf{p}_d]_2 & \dots & [\mathbf{p}_d]_d & 0 & 1 \end{vmatrix} \tag{A.5}$$

Multiplying the previous to the determinant of V_Q gives

$$\left|V_P^T V_Q\right| = \begin{vmatrix} \mathbf{p}_0^T\mathbf{q}_0 & \mathbf{p}_0^T\mathbf{q}_1 & \dots & \mathbf{p}_0^T\mathbf{q}_d & 1 \\ \mathbf{p}_1^T\mathbf{q}_0 & \mathbf{p}_1^T\mathbf{q}_1 & \dots & \mathbf{p}_1^T\mathbf{q}_d & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{p}_{d-1}^T\mathbf{q}_0 & \mathbf{p}_{d-1}^T\mathbf{q}_1 & \dots & \mathbf{p}_{d-1}^T\mathbf{q}_d & 1 \\ 1 & 1 & \dots & 1 & 0 \\ \mathbf{p}_d^T\mathbf{q}_0 & \mathbf{p}_d^T\mathbf{q}_1 & \dots & \mathbf{p}_d^T\mathbf{q}_d & 1 \end{vmatrix} \tag{A.6}$$

Interchanging the last two rows returns:

$$\begin{aligned} \left|V_P^T V_Q\right| &= -\begin{vmatrix} \mathbf{p}_0^T\mathbf{q}_0 & \mathbf{p}_0^T\mathbf{q}_1 & \dots & \mathbf{p}_0^T\mathbf{q}_d & 1 \\ \mathbf{p}_1^T\mathbf{q}_0 & \mathbf{p}_1^T\mathbf{q}_1 & \dots & \mathbf{p}_1^T\mathbf{q}_d & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{p}_d^T\mathbf{q}_0 & \mathbf{p}_d^T\mathbf{q}_1 & \dots & \mathbf{p}_d^T\mathbf{q}_d & 1 \\ 1 & 1 & \dots & 1 & 0 \end{vmatrix} \qquad (A.7) \\ &= -\begin{vmatrix} P^TQ & \mathbb{1} \\ \mathbb{1}^T & 0 \end{vmatrix} \\ &= -\left|M\right| \end{aligned}$$

Let C be the matrix inside the Cayley-Menger bi-determinant. Using the fact that $d(\mathbf{p}_i, \mathbf{q}_j)^2 = ||\mathbf{p}_i - \mathbf{q}_j||^2 = (\mathbf{p}_i - \mathbf{q}_j)^T(\mathbf{p}_i - \mathbf{q}_j)$, we write

$$\left|C\right| = \begin{vmatrix} 0 & 1 & \dots & 1 \\ 1 & \mathbf{p}_0^T\mathbf{p}_0 + \mathbf{q}_0^T\mathbf{q}_0 - 2\mathbf{p}_0^T\mathbf{q}_0 & \dots & \mathbf{p}_0^T\mathbf{p}_0 + \mathbf{q}_d^T\mathbf{q}_d - 2\mathbf{p}_0^T\mathbf{q}_d \\ 1 & \mathbf{p}_1^T\mathbf{p}_1 + \mathbf{q}_0^T\mathbf{q}_0 - 2\mathbf{p}_1^T\mathbf{q}_0 & \dots & \mathbf{p}_1^T\mathbf{p}_1 + \mathbf{q}_d^T\mathbf{q}_d - 2\mathbf{p}_1^T\mathbf{q}_d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{p}_d^T\mathbf{p}_d + \mathbf{q}_0^T\mathbf{q}_0 - 2\mathbf{p}_d^T\mathbf{q}_0 & \dots & \mathbf{p}_d^T\mathbf{p}_d + \mathbf{q}_d^T\mathbf{q}_d - 2\mathbf{p}_d^T\mathbf{q}_d \end{vmatrix} \tag{A.8}$$

Applying row and column operations

$$Row_i \leftarrow Row_i - \mathbf{p}_{i-2}^T\mathbf{p}_{i-2}Row_1$$

$$Col_j \leftarrow Col_j - \mathbf{q}_{j-2}^T\mathbf{q}_{j-2}Col_1$$

for $2 \leq$ i, j $\leq d+2$, result in:

$$\left|C\right| = -\tfrac{1}{2} \begin{vmatrix} 0 & -2 & \dots & -2 \\ 1 & -2\mathbf{p}_0^T\mathbf{q}_0 & \dots & -2\mathbf{p}_0^T\mathbf{q}_d \\ 1 & -2\mathbf{p}_1^T\mathbf{q}_0 & \dots & -2\mathbf{p}_1^T\mathbf{q}_d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & -2\mathbf{p}_d^T\mathbf{q}_0 & \dots & -2\mathbf{p}_d^T\mathbf{q}_d \end{vmatrix} \tag{A.9}$$

Extracting the repeated scalars and noticing that an even number of permutations is required, one can use equation (A.7), so that

$$\begin{aligned} \left|C\right| &= -\tfrac{(-2)^{d+1}}{2}\left|M\right| \\ &= (-1)^{d+1}2^d\left|V_P^TV_Q\right| \end{aligned} \tag{A.10}$$

Or, as we defined before,

$$\begin{aligned} \left|V_P^T V_Q\right| &= 2\left(-\tfrac{1}{2}\right)^{d+1}\left|C\right| \\ &= D(\mathbf{p}_0,\dots,\mathbf{p}_d;\mathbf{q}_0,\dots,\mathbf{q}_d) \end{aligned} \tag{A.11}$$

Now, using (A.1) and (A.2), we can see that

$$D(\mathcal{P};\mathcal{Q}) = (d!)\text{Vol}(\mathcal{P})\cdot(d!)\text{Vol}(\mathcal{Q}). \tag{A.12}$$

By taking the sets $\mathcal{Q} = \mathcal{P}$ one finds that

$$\begin{aligned} D(\mathcal{P}) &= D(\mathcal{P};\mathcal{P}) \\ &= (d!)^2\text{Vol}(\mathcal{P})^2. \end{aligned} \tag{A.13}$$

Therefore, the Cayley-Menger Bi-determinant is proportional to the product of the signed volumes of the sets of points as previously defined.

A.2. BeagleBoneGreen-Wireless setup documentation

A.2.1. Initialization script and basic Linux configuration

So far, we could not configure a persistent rule to bring our mesh network interface up automatically at boot. We also found that disabling the wlan0 pre-configured while compiling Linux, would disable the wireless card and further `iw` commands would not work.

We want to find how and where the initialization of the wireless wlan0 interface is brought up and make it similar for our mesh0 interface, but due to time constraints we decided to postpone this task.

At this point, the system is working quite reliably with the following configurations and initialization script.

/etc/hostname

```
BBGWyyyyyyxx
```

This file only has the defined hostname for each of the boards. We choose to call each board by the *BBGW* string followed by its serial number *yyyyyyxx*, which should be a unique name for networks formed only by BeagleBoneGreen-Wireless boards.

/etc/hosts

```
127.0.0.1       localhost
127.0.1.1       BBGW160537ZZ    bbgwZZ

192.168.100.XX  BBGW160508XX    bbgwXX
192.168.100.YY  BBGW160519YY    bbgwYY
192.168.100.WW  BBGW160516WW    bbgwWW
192.168.100.WW  BBGW160537ZZ    bbgwZZ

192.168.XX.2    BBGW160508XX    bbgwXX
192.168.YY.2    BBGW160519YY    bbgwYY
192.168.WW.2    BBGW160516WW    bbgwWW
192.168.ZZ.2    BBGW160537ZZ    bbgwZZ

192.168.XX.1    tecchio\*       tecchio
192.168.YY.1    tecchio\*       tecchio
192.168.WW.1    tecchio\*       tecchio
192.168.ZZ.1    tecchio\*       tecchio

# The following lines are desirable for IPv6 capable hosts
```

```
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Here, we define the hosts table for our entire *usb0* and *mesh0* networks. First column is the IP addresses, second is the device name, and further columns are all possible aliases for each device.

BBG-W's names were defined as *BBGWyyyyyyxx* where *yyyyyyxx* is the serial number of each board.

Aliases were created to shorten names, and simplify their use. They consist of *bbgw* string plus *xx*, the last two digits of the serial number.

This way to define aliases and networks were possible due to all used boards having different last two digits for their serial numbers. It is an easy way for human operators to identify and connect to the correct board, as their serial numbers are on the boards.

A more general static network can be enabled by using sequential numbers from 1 to 254, which does not create any problems or disadvantages given that all boards have all correct addresses, names and aliases.

/etc/network/interfaces

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback
```

```
#auto eth0
#iface eth0 inet dhcp

# Ethernet/RNDIS gadget (g_ether)
# ... or on host side, usbnet and random hwaddr
iface usb0 inet static
        address 192.168.XX.2
        netmask 255.255.255.252
        network 192.168.XX.0
        gateway 192.168.XX.1

auto wlan0
iface wlan0 inet dhcp
    pre-up wpa_supplicant -B -Dnl80211 -iwlan0 \
    -c/home/ubuntu/wpa_supplicant.conf
    post-down killall -q wpa_supplicant
```

File explanation:

loopback Standard loopback initialization.

eth0 There is no cable Ethernet interface in this board, but it can be added as an external Capeboard module. That is why it is only commented out.

usb0 USB virtual Ethernet port is created at boot, similar to other BeagleBone boards. We modified the standard addresses here so that we can connect multiple boards to the same computer. Each board creates a subnet 192.168.XX.0, where XX is the last two numbers of its serial number. This subnet has only two possible addresses 192.168.XX.1, computer host and gateway, and 192.168.XX.2, the board itself.

wlan0 We use a pretty standard configuration here with wpa_supplicant. For more information consult our *wpa_supplicant.conf* file. Ubuntu usually use a network manager, which we don't need.

We want to use a similar configuration for our mesh0 interface using a specific file for it *mesh_wpa_supplicant.conf*. However, so far, we were not able to set it in a correct way. We did find some new resources from Texas Instruments with script examples to configure it, after we tried to do this. As it is working for now, we will try this later on.

~/wpa_supplicant.conf

Our basic *wpa_supplicant.conf* file used while testing.

```
network={
   ssid="#####"
   scan_ssid=1
   proto=WPA
   key_mgmt=WPA-PSK
   psk="**********"
}
```

/etc/init.d/iw_mesh_init.sh

The following file works, but performs no tests before issuing commands nor check if such commands were correctly executed with desired outcomes. It is positioned with other root initialization scripts. Maybe, it should run on user local init scripts instead.

Ubuntu recommends the use of *Upstart* [82] . We briefly tried to use it unsuccessfully.

This file adds an interface called mesh0 of type mesh with mesh_id localizationmesh to the physical device phy0, TI's WL1835 wireless module. It sets mesh0 interface channel to 10. Which is a standard 2.4 GHz 20 MHz channel (Non HT). According to TI's documentation for this wireless module, all wireless interfaces should run on the same channel to minimize latency, as the module does not need to switch frequencies each time a new packet from a different channel is queued or it opens for packet reception.

Later, it configures IP address, netmask and broadcast address for mesh0 interface. Again, the subnet is configured as 192.168.100.0, while each board uses its configured IP address listed on */etc/hosts* for this subnet.

Finally, it adds a monitor interface, mon0, to mesh0 interface and brings it up.

```
 #!/bin/bash


sudo iw phy0 interface add mesh0 type mp mesh_id localizationmesh
sudo iw mesh0 set channel 10
sudo ifconfig mesh0 192.168.100.XX netmask 255.255.255.0 \
     broadcast 192.168.100.255
sudo iw mesh0 interface add mon0 type monitor
sudo ifconfig mon0 up
```

Current drawbacks and future work

The major drawback of our current configuration is security. We do not have an authentication method nor do we have any mac address filter. Therefore, our mesh network is open, any device that knows the subnet configuration and SSID can enter this network.

We also did not define a default gateway for our mesh network, so devices that are not simultaneously connected to another access point (AP), do not have an Internet connection. One way to correct these is to configure one node as a gateway and dhcp server for other devices connected on our mesh. We intend to follow some example scripts from Texas Instruments in order to set a dhcp server using *isc-dhcp-server*, *wpa_supplicant*, *SAE authentication* and `bridge`.

Another point we would like to correct is related to MAC addresses. Each board has a MAC address and serial number printed on it, but they seem to not be available in any physical memory onboard. Meanwhile, Texas Instruments hard codes a MAC address in its modules. The interest thing is that neither are used by the compiled Linux kernel for all available interfaces.

We investigated a little about setting a mac address in Linux and it is possible. Though we did not try to do it, as we have two wireless interfaces and one board manufacturer given mac address per board. Another point to define is related to make such mac addresses automatically configured in each board at OS install or boot.

Finally, we desire to build a script that automatically identifies each board and sets all configuration files and initialization scripts defined on this section.

A.2.2. Building Linux kernel and TI's driver from repo

We follow Robert Nelson's eewiki.net [75] closely for building our BBGW Linux kernel. After installing the compiled kernel and Root File System on a micro SD card, we follow TI's wikis WiLink8 Linux Getting Started Guide [76] , WL18xx WiFi Build Process [78] , WL18xx System Build Scripts [79] and WL18xx Platform Integration Guide [77] to compile and install the drivers on the micro SD card.

We use the path `~/code/bbgw` as the root path for all operations in our host computer used to build these files. The host computer uses Ubuntu 14.04 LTS.

Linux kernel and Root File System

First go to our root path:

```
mkdir ~/code/bbgw
cd ~/code/bbgw
```

ARM Cross Compiler: GCC

Pre-built x64 version of Linaro GCC for generic Linux:

```
wget -c https://releases.linaro.org/components/toolchain/binaries/\
5.3-2016.02/arm-linux-gnueabihf/gcc-linaro-5.3-2016.02-x86_64_arm-\
linux-gnueabihf.tar.xz


tar xf gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.xz
export CC=‘pwd‘/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/\
bin/arm-linux-gnueabihf-
```

Test:

```
${CC}gcc --version
arm-linux-gnueabihf-gcc (Linaro GCC 5.3-2016.02) 5.3.1 20160113
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS \
FOR A PARTICULAR PURPOSE.
```

Bootloader: U-Boot

Das U-Boot - the Universal Boot Loader [83]:

```
git clone https://github.com/u-boot/u-boot
cd u-boot/
git checkout v2016.03 -b tmp
```

eewiki.net patches [84]:

```
cd u-boot
```

```
wget -c https://rcn-ee.com/repos/git/u-boot-patches/v2016.03/0001\
-am335x_evm-uEnv.txt-bootz-n-fixes.patch
```

```
patch -p1 < 0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch
```

Still at `/u-boot` configure and compile:

```
make ARCH=arm CROSS_COMPILE=${CC} distclean
make ARCH=arm CROSS_COMPILE=${CC} am335x_evm_defconfig
make ARCH=arm CROSS_COMPILE=${CC}
```

Linux kernel

The given script will build the kernel, modules, device trees binaries and copy them to the deploy directory. Downloading source files for am33x-v4.6, current stable release:

```
cd ~/code/bbgw
git clone https://github.com/RobertCNelson/bb-kernel
cd bb-kernel/
git checkout origin/am33x-v4.6 -b tmp
```

Build:

```
./build_kernel.sh
```

At this point, we will be directed to a configuration set up in order to choose which modules will be built. We need to check WL18xx Platform Integration Guide [77].

First make sure that CFG80211 and MAC80211 are not built at all.:

```
CONFIG_CFG80211=N
CONFIG_MAC80211=N
```

We also need to verify that all of the following options are enabled in the .config file.:

```
#
# Network testing
#

# MAC 80211
CONFIG_WLAN=y
CONFIG_WIRELESS=y
CONFIG_WIRELESS_EXT=y
CONFIG_WL12XX_PLATFORM_DATA=y

CONFIG_KEYS=y
CONFIG_SECURITY=y
CONFIG_CRYPTO=y

CONFIG_CRYPTO_ARC4=y
CONFIG_CRYPTO_ECB=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_MICHAEL_MIC=y
CONFIG_CRYPTO_RNG=y
CONFIG_CRYPTO_AEAD=y
CONFIG_CRYPTO_CCM=y
CONFIG_CRYPTO_GCM=y

CONFIG_RFKILL=y

CONFIG_REGULATOR_FIXED_VOLTAGE=y
CONFIG_CRC7=y
```

```
# The following are needed for soft AP
CONFIG_NETFILTER=y
CONFIG_NETFILTER_ADVANCED=y
CONFIG_NF_CONNTRACK=y
CONFIG_NETFILTER_XTABLES=y
CONFIG_NF_DEFRAG_IPV4=y
CONFIG_NF_CONNTRACK_IPV4=y
CONFIG_NF_CONNTRACK_PROC_COMPAT=y
CONFIG_IP_NF_IPTABLES=y
CONFIG_IP_NF_FILTER=y
CONFIG_IP_NF_TARGET_LOG=y
CONFIG_NF_NAT=y
CONFIG_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=y

CONFIG_INPUT_UINPUT=y

# Enable Ethernet-WLAN Bridge
CONFIG_NETFILTER=y
CONFIG_NETFILTER_ADVANCED=y
CONFIG_BRIDGE_NETFILTER=y
CONFIG_STP=y
CONFIG_BRIDGE=y
CONFIG_BRIDGE_IGMP_SNOOPING=y
CONFIG_LLC=y
CONFIG_INPUT_UINPUT=y
CONFIG_HAS_IOMEM=y
```

```
CONFIG_HAS_IOPORT=y
CONFIG_HAS_DMA=y
CONFIG_NLATTR=y
CONFIG_AVERAGE=y
```

While choosing modules to build, we tried to select all these options as needed. We also enable options related to Bluetooth. Texas Intruments provide a script named *verify_kernel_config.sh* at their System Build Scripts [79] which is also found at their repo [78].

Verify if all settings are correct:

```
verify_kernel_config.sh ~/code/bbgw/bb-kernel
```

Not sure if we used any of the specific options for it, like *BT_HCI*. We actually did not follow any of the other configurations at WL18xx Platform Integration Guide [77].

As we are using a script to build the kernel, we actually built an entire kernel by choosing manually the options and had to rebuilt it again after running the *verify_kernel_config.sh*.

Root File System

As we are going to run ROS, we chose to built Ubuntu 16.04 LTS Root File System. Its standard login:password is ubuntu:temppwd.

Downloading:

```
cd ~/code/bbgw
wget -c https://rcn-ee.com/rootfs/eewiki/minfs/ubuntu-16.04-minimal-armhf
-2016-06-06.tar.xz
```

Verifying:

```
sha256sum ubuntu-16.04-minimal-armhf-2016-06-06.tar.xz
```

```
7fa17e1ceaf3778080f834d51dba5bfd1b0cfb6ccb2f979782dedfd8379db4b3
ubuntu-16.04-minimal-armhf-2016-06-06.tar.xz
```

Extracting:

```
tar xf ubuntu-16.04-minimal-armhf-2016-06-06.tar.xz
```

Setup micro SD card

With micro SD card inserted, use `lsblk` to determine its id, it should be something similar to `/dev/mmcblk0`. Then:

```
export DISK=/dev/mmcblk0
```

Erase partition table/labels on microSD card:

```
sudo dd if=/dev/zero of=${DISK} bs=1M count=10
```

Install Bootloader:

```
sudo dd if=./u-boot/MLO of=${DISK} count=1 seek=1 bs=128k
sudo dd if=./u-boot/u-boot.img of=${DISK} count=2 seek=1 bs=384k
```

Create new partition, checking `sfdisk` version:

```
sudo sfdisk --version
```

If version >= 2.26.x:

```
sudo sfdisk ${DISK} <<-__EOF__
1M,,L,*
__EOF__
```

Else:

```
sudo sfdisk --unit M ${DISK} <<-__EOF__
1,,L,*
__EOF__
```

Format new partition, checking `mkfs.ext4` version:

```
sudo mkfs.ext4 -V
```

If version >= 1.43:

```
for: DISK=/dev/mmcblk0
sudo mkfs.ext4 -L rootfs -O ^metadata_csum,^64bit ${DISK}p1

for: DISK=/dev/sdX
sudo mkfs.ext4 -L rootfs -O ^metadata_csum,^64bit ${DISK}1
```

Else:

```
for: DISK=/dev/mmcblk0
sudo mkfs.ext4 -L rootfs ${DISK}p1

for: DISK=/dev/sdX
sudo mkfs.ext4 -L rootfs ${DISK}1
```

Mounting partition:

```
sudo mkdir -p /media/rootfs/

for: DISK=/dev/mmcblk0
sudo mount ${DISK}p1 /media/rootfs/

for: DISK=/dev/sdX
```

```
sudo mount ${DISK}1 /media/rootfs/
```

Backing up the Bootloader:

```
sudo mkdir -p /media/rootfs/opt/backup/uboot/

sudo cp -v ./u-boot/MLO /media/rootfs/opt/backup/uboot/

sudo cp -v ./u-boot/u-boot.img /media/rootfs/opt/backup/uboot/
```

Fix for possible older bootloader in the eMMC:

```
sudo nano /media/rootfs/uEnv.text
```

and copy the following inside it:

```
##This will work with: Angstrom's 2013.06.20 u-boot.


loadaddr=0x82000000

fdtaddr=0x88000000

rdaddr=0x88080000


initrd_high=0xffffffff

fdt_high=0xffffffff


#for single partitions:

mmcroot=/dev/mmcblk0p1


loadximage=load mmc 0:1 ${loadaddr} /boot/vmlinuz-${uname_r}

loadxfdt=load mmc 0:1 ${fdtaddr} /boot/dtbs/${uname_r}/${fdtfile}

loadxrd=load mmc 0:1 ${rdaddr} /boot/initrd.img-${uname_r}; setenv\

rdsize ${filesize}

loaduEnvtxt=load mmc 0:1 ${loadaddr} /boot/uEnv.txt ; env import -t\
${loadaddr} ${filesize};
loadall=run loaduEnvtxt; run loadximage; run loadxfdt;

mmcargs=setenv bootargs console=tty0 console=${console} ${optargs}\
${cape_disable} ${cape_enable} root=${mmcroot} rootfstype=\
${mmcrootfstype}${cmdline}

uenvcmd=run loadall; run mmcargs; bootz ${loadaddr} - ${fdtaddr};
```

Installing Kernel and Root File System

The build scripts will output the *kernel_version=4.X.Y-Z* which we need to export as a variable. The output is like:

```
-----------------------------
Script Complete
eewiki.net: [user@localhost:~$ export kernel_version=4.X.Y-Z]
-----------------------------
```

It should be *kernel_version=4.6.3-bone3*. So:

```
export kernel_version=4.X.Y-Z
```

Copy the Root File System:

```
sudo tar xfvp ./*-*-*-armhf-*/armhf-rootfs-*.tar -C /media/rootfs/
```

Set uname_r in /boot/uEnv.txt:

```
sudo sh -c "echo 'uname_r=${kernel_version}' >> /media/rootfs/\
boot/uEnv.txt"
```

Copy kernel image:

```
sudo cp -v ./bb-kernel/deploy/${kernel_version}.zImage \
/media/rootfs/boot/vmlinuz-${kernel_version}
```

Copy kernel device tree binaries:

```
sudo mkdir -p /media/rootfs/boot/dtbs/${kernel_version}/
sudo tar xfv ./bb-kernel/deploy/${kernel_version}-dtbs.tar.gz -C \
/media/rootfs/boot/dtbs/${kernel_version}/
```

Copy kernel modules:

```
sudo tar xfv ./bb-kernel/deploy/${kernel_version}-modules.tar.gz -C\
/media/rootfs/
```

File Systems Table:

```
sudo sh -c "echo '/dev/mmcblk0p1  /  auto  errors=remount-ro  0  1'
>> /media/rootfs/etc/fstab"
```

Network configuration

To run the Ethernet over USB:

```
mkdir ~/code/bbgw/usb_ethernet
cd ~/code/bbgw/usb_ethernet
wget -c https://raw.github.com/RobertCNelson/tools/master/scripts/\
beaglebone-black-g-ether-load.sh
chmod +x beaglebone-black-g-ether-load.sh
```

Copy the file to the micro SD:

```
mkdir /media/rootfs/home/ubuntu/usb_ethernet
cp beaglebone-black-g-ether-load.sh /media/rootfs/home/ubuntu/usb_ethernet/.
```

While running the built Linux:

```
sudo apt-get install udhcpd


cd ~/usb_ethernet

sudo ./beaglebone-black-g-ether-load.sh
```

Copying micro SD files to eMMC

Downloading the script:

```
cd ~/code/bbgw

mkdir uSD_eMMC

cd uSD_eMMC

wget https://raw.githubusercontent.com/RobertCNelson/boot-scripts/\

master/tools/eMMC/bbb-eMMC-flasher-eewiki-ext4.sh

chmod +x bbb-eMMC-flasher-eewiki-ext4.sh
```

Then, copy it to the micro SD:

```
mkdir /media/rootfs/home/ubuntu/eMMC

cp bbb-eMMC-flasher-eewiki-ext4.sh /media/rootfs/home/ubuntu/eMMC/.
```

Later, when we need to copy the contents, run:

```
cd ~/eMMC

sudo /bin/bash ./bbb-eMMC-flasher-eewiki-ext4.sh
```

TI's wireless WL18xx module driver

Remove the micro SD card

```
sync
```

```
sudo umount /media/rootfs
```

A.2.3. Capturing RSSI values

We investigate possible ways for capturing RSSI values:

1. Obtaining it directly from TI's driver
2. Using `iw`
3. Using a low level capture library for C or Python like PCAP
4. Using an Ethernet packet capture tool like Wireshark, Tshark or TCPDump with console or file output

We could not find how to extract this data directly from wireless card driver. This should be possible, but maybe not advisable. `iw`, Ubuntu's wireless control software, is not built for this purpose. It has the capability to output RSSI average values, but it is not fast enough nor reliable for logging purposes.

We believe that the best long-term approach is to use a low level library like libpcap in order to capture RSSI values directly from each packet in our future ROS package. C/C++ code should be faster for this purpose.

However, due to implementation time constraints to obtain experimental data for our first test, we decided to use TCPDump. Wireshark requires a graphical interface which is not available. Tshark, Wireshark's console version, should work as well, but was not tested.

Installing TCPDump

To install it run:

```
sudo apt-get install tcpdump
```

Configuring TCPDump

In order to capture an interface with TCPDump, we either use `sudo` or do the following steps, based on Peter Nixon's post at [85]: Creating a specific group for Net administration:

```
sudo groupadd pcap

sudo usermod -a -G pcap ubuntu
```

Changing `tcpdump` group and its permissions:

```
sudo chgrp pcap /usr/sbin/tcpdump

sudo chmod 750 /usr/sbin/tcpdump
```

Giving `tcpdump` net administration rights:

```
sudo setcap cap\_net\_raw,cap\_net\_admin=eip /usr/sbin/tcpdump
```

Rebooting the system:

```
sudo reboot now
```

Checking if the configuration is up:

```
getcap /usr/sbin/tcpdump
```

If the output is equal to:

```
/usr/sbin/tcpdump = cap_net_admin,cap_net_raw+eip
```

Then, it is OK and `tcpdump` should work without `sudo`. In one of the boards it did not keep the `setcap` setting, so we repeated the previous steps and it worked at some point. We are not sure why it would not work.

At this point, if we try running ROS' `roscore`, we get a library linking error. To solve it, do the following:

```
sudo ldconfig /opt/ros/indigo/lib
```

Creating a monitor interface

Using `iw`, we can create a monitor interface which will copy all traffic from its source interface, so that a capture program can be applied to it without major interference:

```
sudo iw dev mesh0 interface add mon0 type monitor
sudo ifconfig mon0 up
```

Where **mesh0** is the name of the source interface, in our case it refers to our Wireless 802.11s interface, and **mon0** is the name of our new monitor interface. For more information, consult *iw's man* []. page We do this step automatically during Linux initialization as explained in A.2.1.

Using TCPDump

Our idea is to constantly capture packets with RSSI values. After some experimentation, we observed that management beacon packets were broadcasted from each board at regular intervals, around 100 ms, and had RSSI values attached.

Therefore, we created a sequence of filters to capture beacon packets and save the console output to a .txt file:

```
tcpdump -imon0 -#en -tttt -Zubuntu type mgt subtype beacon and  ether host\
"00:00:00:00:00:00 or 00:00:00:00:00:00 or 00:00:00:00:00:00 or \
00:00:00:00:00:00" -Qin -c150 > ~/tests/rssi/\
tcpdump_beacon_192.168.100.IP_test_XX.txt
```

Used filters:

-imon0 Selects the monitor interface mon0

-# Prints captured packet numbers

-e Prints lower level information

-n Supress search for hostnames and ip's

-tttt Prints date and time

-Zubuntu After capturing the interface, releases admin rights and switch to user ubuntu

-Qin Selects inbound packets only

-c150 Captures 150 packets, around 50 packets from each other board in the experiment

type mgt Selects management packets

subtype beacon Selects beacon packets

ether host Specifies mac addresses from host boards

For more information regarding TCPDump options and pcap filters consult *tcpdump man pages* [86] and *pcap filters man pages* [87].

BIBLIOGRAPHY

[1] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K. Wang, "Review of internet of things (iot) in electric power and energy systems," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 847–870, April 2018.

[2] K. Routh and T. Pal, "A survey on technological, business and societal aspects of internet of things by q3, 2017," in *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, Feb 2018, pp. 1–4.

[3] B. Ayyappan and P. M. Kumar, "Vehicular ad hoc networks (vanet): Architectures, methodologies and design issues," in *2016 Second International Conference on Science Technology Engineering and Management (ICONSTEM)*, March 2016, pp. 177–180.

[4] R. Hussain, J. Son, H. Eun, S. Kim, and H. Oh, "Rethinking vehicular communications: Merging vanet with cloud computing," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec 2012, pp. 606–609.

[5] G. Song, Y. Zhou, W. Zhang, and A. Song, "A multi-interface gateway architecture for home automation networks," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1110–1113, August 2008.

[6] J. Zhang, G. Song, H. Wang, and T. Meng, "Design of a wireless sensor network based monitoring system for home automation," in *2011 International Conference on Future Computer Sciences and Application*, June 2011, pp. 57–60.

[7] N. Vikram, K. S. Harish, M. S. Nihaal, R. Umesh, A. Shetty, and A. Kumar, "A low cost home automation system using wi-fi based wireless sensor network incorporating internet of things (iot)," in *2017 IEEE 7th International Advance Computing Conference (IACC)*, Jan 2017, pp. 174–178.

[8] F. Thomas and L. Ros, "Revisiting trilateration for robot localization," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 93–101, Feb 2005.

[9] J. Biswas, B. Coltin, and M. Veloso, "Corrective gradient refinement for mobile robot localization," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 73–78.

[10] J. Biswas and M. Veloso, "Wifi localization and navigation for autonomous indoor mobile robots," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 4379–4384.

[11] N. Kim, S. Hong, K. Sung, G. Yu, and J. Seo, "A case study on risk assessment for personal care robot (mobile servant robot)," in *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, Oct 2018, pp. 343–347.

[12] C. Datta, P. Tiwari, H. Y. Yang, I. Kuo, E. Broadbent, and B. A. MacDonald, “An interactive robot for reminding medication to older people,” in *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Nov 2012, pp. 190–190.

[13] S. Huang, T. Tanioka, R. Locsin, M. Parker, and O. Masory, “Functions of a caring robot in nursing,” in *2011 7th International Conference on Natural Language Processing and Knowledge Engineering*, Nov 2011, pp. 425–429.

[14] SURELINE WIRING, “Home automation,” http://surelinewiring.com/wp-content/uploads/2016/04/home-automation.png.

[15] AskMen.com, “Smart home automation for beginners,” https://images.askmen.com/1080x540/2018/06/18-025641-smart_home_automation_for_beginners.jpg.

[16] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, “Connected vehicles: Solutions and challenges,” *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 289–299, Aug 2014.

[17] X. Yang, L. Liu, N. H. Vaidya, and F. Zhao, “A vehicle-to-vehicle communication protocol for cooperative collision warning,” in *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.*, Aug 2004, pp. 114–123.

[18] K. Zheng, Q. Zheng, P. Chatzimisios, W. Xiang, and Y. Zhou, “Heterogeneous vehicular networking: A survey on architecture, challenges, and solutions,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2377–2396, Fourthquarter 2015.

[19] S. Biswas, R. Tatchikou, and F. Dion, “Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety,” *IEEE Communications Magazine*, vol. 44, no. 1, pp. 74–82, Jan 2006.

[20] H. Hartenstein and L. P. Laberteaux, “A tutorial survey on vehicular ad hoc networks,” *IEEE Communications Magazine*, vol. 46, no. 6, pp. 164–171, June 2008.

[21] Qualcomm, “Qualcomm v2x header,” https://icdn5.digitaltrends.com/image/qualcomm-v2x-header-2000x1125.jpg.

[22] Compare.com, “Vehicle to vehicle,” https://www.compare.com/wp-content/uploads/2017/09/vehicle-to-vehicle.jpg.

[23] Unknown, “Undersea robot exploration,” https://s3-us-west-2.amazonaws.com/uw-s3-cdn/wp-content/uploads/sites/6/2017/11/04133512/DOEdyNgX4AEe1BY.jpg-large.jpg.

[24] PSA Security Network, “Robotic assistance devices psa,” https://security-images.scdn2.secure.raxcdn.com/news/g5_image/Robotic-Assistance-Devices-PSA-920.jpg.

[25] B. Q. Ferreira, J. Gomes, C. Soares, and J. P. Costeira, “Collaborative localization of vehicle formations based on ranges and bearings,” in *2016 IEEE Third Underwater Communications and Networking Conference (UComms)*, Aug 2016, pp. 1–5.

[26] T. G. Kim, H. Choi, and N. Y. Ko, “Localization of a robot using particle filter with range and bearing information,” in *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Oct 2013, pp. 368–370.

[27] N. Atanasov, R. Tron, V. M. Preciado, and G. J. Pappas, “Joint estimation and localization in sensor networks,” in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 6875–6882.

[28] I. Shames, A. N. Bishop, and B. D. O. Anderson, “Analysis of noisy bearing-only network localization,” *IEEE Transactions on Automatic Control*, vol. 58, no. 1, pp. 247–252, Jan 2013.

[29] R. Tron and K. Daniilidis, “An optimization approach to bearing-only visual homing with applications to a 2-d unicycle model,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4235–4242.

[30] R. Tron, J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, “A Distributed Optimization Framework for Localization and Formation Control: Applications to Vision-Based Measurements,” *IEEE Control Systems Magazine*, vol. 36, no. 4, pp. 22–44, 2016.

[31] E. Bjørne, T. A. Johansen, and E. F. Brekke, “Redesign and analysis of globally asymptotically stable bearing only slam,” in *International Conference on Information Fusion (Fusion)*, July 2017, pp. 1–8.

[32] U. A. Khan, S. Kar, and J. M. F. Moura, “Distributed Sensor Localization in Random Environments Using Minimal Number of Anchor Nodes,” *Trans. Sig. Proc.*, vol. 57, no. 5, pp. 2000–2016, 2009.

[33] Y. Diao, Z. Lin, and M. Fu, “A barycentric coordinate based distributed localization algorithm for sensor networks,” *IEEE Transactions on Signal Processing*, vol. 62, no. 18, pp. 4760–4771, Sept 2014.

[34] U. Khan, S. Kar, and J. Moura, “DILAND: An Algorithm for Distributed Sensor Localization With Noisy Distance Measurements,” *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1940–1947, 2010.

[35] T. Han, Z. Lin, R. Zheng, Z. Han, and H. Zhang, “A barycentric coordinate based approach to three-dimensional distributed localization for wireless sensor networks,” in *2017 13th IEEE International Conference on Control Automation (ICCA)*, July 2017, pp. 600–605.

[36] J. Aspnes, T. Eren, D. K. Goldenberg, A. S. Morse, W. Whiteley, Y. R. Yang, B. D. O. Anderson, and P. N. Belhumeur, “A theory of network localization,” *IEEE Transactions on Mobile Computing*, vol. 5, no. 12, pp. 1663–1678, 2006.

[37] J. Smith and J. Abel, “Closed-form least-squares source location estimation from range-difference measurements,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 12, pp. 1661–1669, December 1987.

[38] S. Pandey and S. Varma, “A range based localization system in multihop wireless sensor networks: A distributed cooperative approach,” *Wireless Personal Communications*, vol. 86, no. 2, pp. 615–634, Jan 2016. [Online]. Available: https://doi.org/10.1007/s11277-015-2948-3

[39] Yi Shang and W. Ruml, “Improved mds-based localization,” in *IEEE INFOCOM 2004*, vol. 4, March 2004, pp. 2640–2651 vol.4.

[40] H. Shao, X. Zhang, and Z. Wang, “Efficient closed-form algorithms for aoa based self-localization of sensor nodes using auxiliary variables,” *IEEE Transactions on Signal Processing*, vol. 62, no. 10, pp. 2580–2594, May 2014.

[41] P. Kristalina, A. Pratiarso, T. Badriyah, and Erik Dwi Putro, “A wireless sensor networks localization using geometric triangulation scheme for object tracking in urban search and rescue application,” in *2016 2nd International Conference on Science in Information Technology (ICSITech)*, Oct 2016, pp. 254–259.

[42] A. V. Tondwalkar and P. Vinayakray-Jani, “Terrestrial localization by using angle of arrival measurements in wireless sensor network,” in *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, Dec 2015, pp. 188–191.

[43] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller, “Anchor-free distributed localization in sensor networks,” in *International Conference on Embedded Networked Sensor Systems*. ACM, 2003, pp. 340–341.

[44] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar, “Distributed localization of networked cameras,” in *International Conference on Information Processing in Sensor Networks*, 2006, pp. 34–42.

[45] A. Barton-Sweeney, D. Lymberopoulos, and A. Savvides, “Sensor Localization and Camera Calibration in Distributed Camera Sensor Networks,” in *International Conference on Broadband Communications, Networks and Systems*, 2006.

[46] L. Xiao, S. Boyd, and S.-J. Kim, “Distributed average consensus with least-mean-square deviation,” *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, 2007.

[47] S. Shahrampour and A. Jadbabaie, “Distributed online optimization in dynamic environments using mirror descent,” *IEEE Transactions on Automatic Control*, vol. 63, no. 3, pp. 714–725, March 2018.

[48] S. Safavi, U. A. Khan, S. Kar, and J. M. F. Moura, “Distributed localization: A linear theory,” *Proceedings of the IEEE*, vol. 106, no. 7, pp. 1204–1223, July 2018.

[49] S. Safavi and U. A. Khan, “Localization in mobile networks via virtual convex hulls,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 1, pp. 188–201, March 2018.

[50] U. A. Khan, S. Kar, and J. M. F. Moura, “Distributed sensor localization using barycentric coordinates,” in *2009 3rd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, Dec 2009, pp. 65–68.

[51] U. A. Khan, S. Kar, B. Sinopoli, and J. M. F. Moura, “Distributed sensor localization in euclidean spaces: Dynamic environments,” in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2008, pp. 361–366.

[52] S. Safavi, U. A. Khan, S. Kar, and J. M. F. Moura, “Linear distributed algorithms for localization in mobile networks,” in *2018 IEEE Statistical Signal Processing Workshop (SSP)*, June 2018, pp. 638–642.

[53] U. A. Khan, A. Korniienko, and K. H. Johansson, “An h ∞-based approach for robust sensor localization,” in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 1719–1724.

[54] X. Huang, Y. Tian, and B. Wang, “A distributed localization algorithm based on barycentric coordinate with communication delays and package loss,” in *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 8119–8124.

[55] S. Capkun, M. Hamdi, and J. . Hubaux, “Gps-free positioning in mobile ad-hoc networks,” in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Jan 2001, pp. 10 pp.–.

[56] A. T. Ihler, J. W. Fisher, R. L. Moses, and A. S. Willsky, “Nonparametric belief propagation for self-localization of sensor networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 809–819, April 2005.

[57] B. H. Cheng, R. E. Hudson, F. Lorenzelli, L. Vandenberghe, and K. Yao, “Distributed gauss-newton method for node loclaization in wireless sensor networks,” in *IEEE 6th Workshop on Signal Processing Advances in Wireless Communications, 2005.*, June 2005, pp. 915–919.

[58] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, “Instrumenting the world with wireless sensor networks,” in *2001 IEEE International Conference on Acoustics, Speech, and*

Signal Processing. Proceedings (Cat. No.01CH37221), vol. 4, May 2001, pp. 2033–2036 vol.4.

[59] G. Oliva, S. Panzieri, F. Pascucci, and R. Setola, "Sensor networks localization: Extending trilateration via shadow edges," *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2752–2755, 2015.

[60] H. Li and F. Nashashibi, "Cooperative multi-vehicle localization using split covariance intersection filter," in *2012 IEEE Intelligent Vehicles Symposium*, June 2012, pp. 211–216.

[61] Y. Zhou, Jun Li, and L. Lamont, "Multilateration localization in the presence of anchor location uncertainties," in *2012 IEEE Global Communications Conference (GLOBECOM)*, Dec 2012, pp. 309–314.

[62] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, "Localization from mere connectivity," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, ser. MobiHoc '03. New York, NY, USA: ACM, 2003, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/778415.778439

[63] C. D. Franco, A. Prorok, N. Atanasov, B. Kempke, P. Dutta, V. Kumar, and G. J. Pappas, "Calibration-Free Network Localization Using Non-line-of-sight Ultra-wideband Measurements," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2017, pp. 235–246.

[64] J. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, 1964.

[65] L. Blumenthal, *Theory and applications of distance geometry*. Chelsea Pub. Co., 1970.

[66] M. Berger, M. Cole, and S. Levy, *Geometry I*, ser. Universitext. Springer Berlin Heidelberg, 2009.

[67] J. Barzilai and J. Borwein, "Two-Point Step Size Gradient Methods," *IMA Journal of Numerical Analysis*, vol. 8, no. 1, pp. 141–148, 1988.

[68] A. Johnson, "Error ellipse," https://www.mathworks.com/matlabcentral/fileexchange/4705-error-ellipse, July 2015.

[69] Canonical Ltd., "ubuntu," https://ubuntu.com/about.

[70] O. S. R. Foundation, "Ros - robotic operating system," http://www.ros.org.

[71] B. Foundation, "Beagleboard," https://beagleboard.org/bone.

[72] L. HARDKERNEL CO., "Odroid," https://www.hardkernel.com/.

[73] R. P. FOUNDATION, “Raspeberry pi,” https://www.raspberrypi.org/.

[74] SeeedStudio, “Beaglebonegreen - wireless,” https://www.seeedstudio.com/category/Development-Platforms-c-1002/single-board-computer-c-950/beaglebone-c-954/BeagleBone-Green-Wireless-Development-Board-TI-AM335x-WiFi-BT.html.

[75] R. Nelson, “eewiki linux on arm,” https://www.digikey.com/eewiki/display/linuxonarm/BeagleBone+Black.

[76] Texas Instruments, “Wilink8 linux getting started guide,” http://processors.wiki.ti.com/index.php/WiLink8_Linux_Getting_Started_Guide.

[77] ——, “Wl18xx platform integration guide,” http://processors.wiki.ti.com/index.php/WL18xx_Platform_Integration_Guide.

[78] ——, “Wilink8 wlan,” http://git.ti.com/wilink8-wlan/build-utilites/blobs/master/verify_kernel_config.sh.

[79] ——, “Wl18xx system build scripts,” http://processors.wiki.ti.com/index.php/WL18xx_System_Build_Scripts.

[80] B. Kempke, P. Pannuto, and P. Dutta, “Polypoint: Guiding indoor quadrotors with ultra-wideband localization,” in *Proceedings of the 2nd International Workshop on Hot Topics in Wireless*, September 2015, paper, pp. 16–20.

[81] A. Green and J. Berg, “Radiotap,” http://www.radiotap.org/.

[82] Canonical Ltd., “Upstart,” http://upstart.ubuntu.com/.

[83] DENX Software Engineering, “U-boot,” http://www.denx.de/wiki/U-Boot.

[84] R. Nelson, “u-boot-patches,” https://github.com/eewiki/u-boot-patches.

[85] P. Nixon, “Configure tcpdump to work as non-root user on opensuse using file system capabilities,” https://peternixon.net/news/2012/01/28/configure-tcpdump-work-non-root-user-opensuse-using-file-system-capabilities/.

[86] The Tcpdump Group, “Tcpdump,” http://www.tcpdump.org/manpages/tcpdump.1.html.

[87] ——, “pcap,” http://www.tcpdump.org/manpages/pcap-filter.7.html.