Nesterov Accelerated ADMM for Distributed Pose Graph Optimization in SLAM problems

L. Bosland

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Nesterov Accelerated ADMM for Distributed Pose Graph Optimization in SLAM problems

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

L. Bosland

January 28, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright ©
All rights reserved.

Abstract

A common problem in robotics is the simultaneous localization and mapping (SLAM) problem. Here, a robot needs to create a map of its surroundings while simultaneously localizing itself in this map. An unknown environment is assumed. Traditionally, it has been approached through filtering solutions. This paradigm has shifted to pose graph optimization (PGO). This method scales well with large maps and is fast and accurate. Furthermore, it is especially suited to the distributed SLAM problem as existing distributed optimization methods can be leveraged. One such method is the alternating direction method of multipliers (ADMM), which has been used in distributed PGO. ADMM has a simple implementation and can achieve high accuracy in distributed PGO. A solution of good quality can be acquired in a few iterations with ADMM.

However, ADMM is slow to converge to high accuracy. This thesis introduces an algorithm which implements Nesterov acceleration in the ADMM algorithm for distributed PGO in SLAM problems. Such an implementation will be novel. To create the proposed Nesterov accelerated ADMM (N-ADMM) algorithm, the current literature is adapted and extended based on the choices made in this thesis. The main research question is how to make these choices.

The proposed N-ADMM algorithm is implemented in `C++` and compared with unaccelerated ADMM and the state of the art. N-ADMM has shown better performance in some scenarios. To further research what these scenarios are characterized by, two models are introduced to create new datasets of which the parameters can be controlled. The effects of graph size and bad initial guesses are investigated.

Table of Contents

List of acronyms v
List of symbols vi

Acknowledgments **vii**

1 Introduction **1**
1.1 State of the art 2
1.2 Research questions 3
1.3 Contributions 4

2 Distributed pose graph optimization (PGO) **7**
2.1 Centralized PGO 8
2.2 Distributed PGO: splitting formulation 10

3 Alternating direction method of multipliers (ADMM) for distributed PGO **13**
3.1 ADMM 14
3.1.1 ADMM iterations 14
3.1.2 Multi-block ADMM 16
3.2 ADMM for distributed PGO 16

4 Nesterov accelerated ADMM (N-ADMM) for distributed PGO **19**
4.1 Acceleration 20
4.2 Stabilizing framework 20
4.3 Proposed algorithm 22

5 Performance of N-ADMM **25**
5.1 Comparison with state of the art 26
5.2 Generated datasets 29
5.2.1 Block model 30
5.2.2 Gaussian model 32

6 Conclusions **35**
6.1 Answers to the research questions 35
6.2 Further research 37
6.3 Contributions 39

Bibliography **41**

A Graphical PGO example **45**

B Supporting results and figures **47**

Glossary

List of acronyms

ADMM	Alternating direction method of multipliers
DGS	Distributed Gauss-Seidel method
HOGMan	Hierarchical optimization on manifolds
AMM	Accelerated majorization-minimization
N-ADMM	Nesterov accelerated alternating direction method of multipliers
PGO	Pose graph optimization
SLAM	Simultaneous localization and mapping
SMF	Small memory footprint method
TSAM2	Tectonic smoothing and mapping

Notation and symbols

$SO(n)$	Special orthogonal group for n dimensions
$SE(n)$	Special Euclidean group for n dimensions
$\mathcal{V}$	Set of nodes corresponding to poses in the pose graph
$\mathcal{E}$	Set of edges corresponding to measurements in the pose graph
$\mathcal{P}$	Set of priors in the pose graph
$\mathcal{S}$	Index set of separators
$\mathcal{G}$	Index set of subgraphs
G	Pose graph
t_i	Translation for pose i
t_{ij}	Relative translation between poses i and j
R_i	Rotation for pose i
R_{ij}	Relative rotation between poses i and j
x_i	Pose i
x_{ij}	Relative pose measurement between i and j
$x^\star$	Set of estimated poses
$\bar{x}_i$	Prior i
x_g	Set of poses belonging to subgraph g
x^s	Separator s
Ω, Φ	Information matrices related to measurement covariance
$\mathscr{L}$	Lagrangian
$\mathscr{L}_\rho$	ρ-augmented Lagrangian
y, λ, ν	Dual variables or Lagrange multipliers for the inequality and equality constraints
u	Scaled dual variables
k	Iteration number
ρ	Penalty factor
σ	Convexity constant
I	Identity matrix of appropriate size
$\mathbf{0}$	Zero vector of appropriate size
$\|\| \cdot \|\|$	2-norm of the argument
$\|\|v\|\|_\Omega^2$	$v^T \Omega v$
$\mathbf{dist}(\cdot, \cdot)$	Distance function

Acknowledgments

I would like to thank my supervisor Tamas Keviczky for his feedback and guidance during the thesis and for his support in the final stage of the Systems and Control master's program. It has always helped me to move in the right direction and to focus on the core of the challenges which presented themselves.

I would also like to thank my family, my good friends and my amazing girlfriend for giving me the motivation to complete this work and for all the encouragements and shared moments along the way.

Lastly, I would like to thank my fellow students for working together and making the long days at university an overall pleasant experience.

Delft, University of Technology L. Bosland
January 28, 2021

Chapter 1

Introduction

Fully autonomous robots must be able to localize themselves in unknown indoor and outdoor environments. Otherwise, it would be impossible for these robots to function in new locations without bumping into their environment or needing the help of a human controller. The problem of creating a map of the environment of the robot and localizing itself within this map has been approached through the *simultaneous localization and mapping* (SLAM) problem [1, 2, 3, 4, 5].

A wide range of applications exist for SLAM which include the following [5]:

- Exploration with aerial, underwater and ground vehicles
- Operations in dangerous environments
- Inspections
- Transportation and distribution
- Autonomous driving

Large areas need to be covered for these applications and the size of the maps have been steadily increasing. For example, the recently emerging fully autonomous cars have to localize themselves and map their surroundings in areas that span cities or even countries. However, when areas grow too large, computations cannot be performed by one unit anymore [6]. A natural solution to this problem is splitting the problem into multiple subproblems and using multiple units for computing. One could use multiple cores of the same processor to solve the subproblems or even use swarms of robots that solve the problem collectively. Data could be exchanged when they pass each other or via a mobile data connection. Another advantage of using multiple units is that the system becomes more robust because the SLAM mission will not be abandoned if a single robot fails [5]. In that case, the task can be taken over by the other units in the system. This is especially useful for hazardous applications, e.g. mine explorations or search and rescue missions in a forest fire. Moreover, if multiple cores of a single processor are used, the computational task can be taken over by the other cores if one core fails. An additional advantage is that data can be collected faster with multiple robots [5]: multiple robots can cover a larger area than a single robot when they are given the same amount of time. Furthermore, accuracy can be improved when multiple robots cover the same area. Therefore, a system consisting of multiple robots is flexible in balancing speed and accuracy. For the reasons above, this thesis will focus on the *distributed* SLAM problem, which allows for the use of multiple robots and cores.

Two paradigms exist in solving SLAM problems: filtering based SLAM and optimization based SLAM [7]. Optimization can be applied in SLAM if the problem is reformulated as a pose graph where poses are denoted as nodes and measurements are denoted as edges in the graph [8]. A graphical example of such a pose graph can be found in Appendix A. The main advantage of using *pose graph optimization* (PGO) is that it scales well with larger maps [7, 9]. Filtering based solutions are limited in this sense because solutions depend on the covariance matrix, which quadratically scales with the size of the map [7]. Therefore, filtering based solutions use more computation time and memory if the area covered by the robots is large. Another advantage of using PGO is that existing optimization methods can be leveraged. This is especially useful for distributed PGO as distributed optimization is an active field of research while a large body of work already exists [10, 11]. Furthermore, while filtering based SLAM can be beneficial when only a small processing budget is available, optimization based SLAM has been shown to be superior in terms of accuracy [9]. Especially because computing power has increased significantly since [9] was written, the advantage of optimization based SLAM is clear. Moreover, optimization based SLAM methods can include outlier rejection techniques, they can handle measurements of different sensors simultaneously and they are fast [12]. They are fast because the sparsity of the problem is exploited. For the reasons above, this thesis will use PGO to solve the SLAM problem.

1.1 State of the art

This report considers the following methods to be the state of the art in distributed PGO:

- Alternating direction method of multipliers (ADMM) [6]
- Accelerated majorization-minimization (AMM) [13]
- Distributed Gauss-Seidel method (DGS) [14]
- Tectonic smoothing and mapping (TSAM2) [15]
- Hierarchical optimization on manifolds (HOGMan) [16]
- Small-memory footprint method (SMF) [17]

Within these methods, ADMM, AMM and DGS are considered to be the closest related works. Some connections will be addressed briefly in this section. These connections will become more clear throughout this thesis.

To formulate the distributed PGO problem two approaches exist: elimination and splitting. TSAM2 [15], HOGMan [16] and SMF [17] use elimination to formulate the problem. In these methods, dense cliques are induced on the separating nodes and bookkeeping of linearization points is necessary [6]. This causes problems when solving large graphs as these methods are computationally expensive and have to rely on linearization [6]. To avoid this, the splitting approach for distributed PGO was introduced in [6]. Moreover, the centralized PGO problem is sparse, which is appealing when performing optimization [7]. This sparsity is preserved when distributed PGO is approached through splitting [6]. Sparsity is not preserved when elimination is used to formulate the problem. Furthermore, the splitting approach connects neatly to the pose graphs, which means the already available factor graph optimizers and classes included in the GTSAM [18] and g^{2}o [19] libraries can be leveraged to solve the subproblems of distributed PGO. For the reasons above, this thesis will use the splitting approach for distributed PGO. Currently, ADMM [6], DGS [14] and AMM [13] approach distributed PGO through splitting.

ADMM [20] has been used in [6] to solve distributed PGO. It has various advantages in distributed optimization. ADMM has a simple implementation and allows for parallelization, which means the subproblems can be solved simultaneously across processing units or robots. Furthermore, minimal information exchange will be required between robots and no linearization is needed when using ADMM [6]. However, while ADMM reaches a solution of acceptable quality in a few iterations, convergence to solutions of high quality is considerably slower than the state of the art. To combat this, parallelization can be applied to distribute computational load. Moreover, ADMM allows for *acceleration*. Since accelerated ADMM has not yet been examined for distributed PGO problems, a clear gap is found in the literature. In this thesis, such an algorithm will be created by combining and extending existing literature.

1.2 Research questions

In this thesis, acceleration will be implemented with the use of Nesterov's method [21] as it has a provably optimal convergence rate of $\mathcal{O}(1/k^2)$ [22, 23]. From the gap in the literature, the research question follows:

"How can Nesterov acceleration be implemented in the ADMM algorithm for solving distributed PGO problems and what adaptations have to be made to the current methods for this implementation?"

The distributed PGO problem will be defined in Chapter 2. The current ADMM solution to distributed PGO will be discussed in Chapter 3. This solution will be extended with Nesterov acceleration in Section 4.1. Then, this accelerated algorithm is combined with a stabilizing framework in Section 4.2. The proposed algorithm is not convergent without this framework. In Chapter 4, the necessary adaptations and extensions of the literature will be discussed. Through these adaptions, the creation of the Nesterov accelerated ADMM (N-ADMM) algorithm is enabled. The proposed N-ADMM algorithm will be introduced in Section 4.3. N-ADMM has been implemented in `C++` to test if it is valid and converges to good solutions. Implementation is done in `C++` to use the existing pose graph optimization libraries GTSAM [18] and g^{2}o [19].

To support the contribution of the N-ADMM algorithm, the following sub-question is formulated:

"How does the performance of N-ADMM compare to the state of the art?"

Performance will be measured by three factors: the number of iterations needed until convergence, the time needed until convergence and the value of the objective function when the algorithm converges. The first two factors define efficiency and the last factor defines accuracy. These factors are also commonly used in the state of the art, which has been defined in Section 1.1. The closest related works are considered to be [6], [14] and [13] as they use the splitting approach to distributed PGO. Further similarities between N-ADMM and the closest related works will be discussed in Chapter 3. To compare N-ADMM to the state of the art, tests will be performed on benchmark datasets in Section 5.1.

The number of publicly available benchmark datasets is limited. While they represent a good mix of scenarios, correlations between certain characteristics of the datasets and the performance of N-ADMM cannot be discerned based on the benchmark datasets alone. The following sub-question arises:

"What characterizes the pose graphs for which optimization with N-ADMM is, relative to the current methods, most efficient?"

In order to find an answer to this question, new datasets have to be created for which the parameters can be varied. Datasets are created based on two new models, which will be introduced in Section 5.2. The programs that produce these datasets have been created in `C++`. To test the influence of graph size, both the block model of Section 5.2.1 and the Gaussian model of Section 5.2.2 are used. Bad initial guesses are modelled with relative pose measurements which have an offset from the trajectory. Only the Gaussian model will be used to test the effect of the mean and variance of bad initial guesses.

1.3 Contributions

By answering the research questions, this thesis will contribute an accelerated ADMM algorithm for solving distributed PGO. It has been shown to be more efficient in select situations. Through the creation of this algorithm, a set of choices was found which produce an effective accelerated multi-block ADMM algorithm. By doing so, a starting point for future research is created and insight was gained for points of improvement. The proposed N-ADMM algorithm provides a better choice for practical applications in distributed PGO. In the worst case scenario, performance is roughly equal to the plain ADMM iterations. However, in the best case scenario, efficiency can be much improved. The main drawback of plain ADMM for distributed PGO is its slow convergence to high accuracy. This thesis contributes a good alternative to the plain ADMM algorithm which can mitigate this drawback significantly. The many advantages of ADMM, e.g. accuracy, parallelization and minimal information exchange, can now be more easily leveraged in practical distributed applications. Furthermore, by creating the N-ADMM algorithm, a multi-block application for the stabilizing framework of [22] is provided.

Furthermore, this thesis compares the performance of the N-ADMM algorithm and the plain ADMM algorithm with the state of the art. Through this comparison, the need for further analysis into the properties of the pose graphs became apparent. While it has been shown that N-ADMM is more efficient in certain scenarios, little is known on the specifics of these scenarios. To investigate this, this thesis contributes two models. Based on these models, datasets with varying characteristics were created. By performing PGO on the resulting datasets, this thesis has investigated the effects of bad initial guesses and graph size. However, no correlation was found for the size of the graph and the relative effectiveness between N-ADMM and plain ADMM. Furthermore, while a correlation was found between poor initial guesses and efficiency, there was no apparent gap between the performance of N-ADMM and ADMM. Further research is necessary to find the exact scenarios in which N-ADMM outperforms ADMM.

In short, this thesis contributes the following:

- Creation of the N-ADMM algorithm by combining and extending current literature.
- Comparison of N-ADMM and ADMM [6] with the state of the art.
- New models for more extensive testing.
- Multi-block application for stabilizing framework [22].

These contributions will be revisited in Chapter 6.

Chapter 2

Distributed pose graph optimization (PGO)

Classically, the simultaneous localization and mapping (SLAM) problem has been approached through filtering based solutions. However, the paradigm has shifted towards optimization based SLAM in recent years. Optimization based SLAM reformulates the problem with the use of pose graphs [8]. Here, a pose graph is constructed by representing the poses as nodes in the graph and the relative pose measurements as edges. Pose graph optimization (PGO) methods have many advantages in solving SLAM. First, they scale well with larger maps [7, 9]. Second, existing optimization methods can be leveraged in PGO. Since distributed optimization is an active field of research and many methods exist [10, 11], it is especially useful to use PGO and leverage the available methods for solving distributed SLAM problems. This thesis considers the distributed problem. Third, PGO has been shown to be more accurate than filtering based SLAM [9]. Lastly, PGO is a fast, general and robust method to solve SLAM [12]. For these reasons, this thesis will use PGO. This thesis will solve the distributed problem for the reasons stated in the introduction and in Section 2.2. For a graphical representation of PGO and the connection with the SLAM problem, the reader is referred to Appendix A.

First, centralized PGO will be discussed in Section 2.1 to introduce the theory. Then, in Section 2.2 a distributed formulation of the PGO problem will be given. This is the separator based splitting formulation [6]. For the elimination formulation of the problem the reader is referred to [15, 16, 17]. This thesis will use the splitting formulation for the reasons stated in Section 2.2. The closest related works [6, 14, 13] also use this formulation.

2.1 Centralized PGO

In this section, the structure of PGO will be discussed. Pose graphs are a special type of factor graph [8] where poses correspond to the nodes $\mathcal{V}$ in the graph. The measurements between the poses are represented by the edges $\mathcal{E}$. Together, they form the directed [24] graph $G = \{\mathcal{V}, \mathcal{E}\}$. For a graphical representation and further information, the reader is referred to Appendix A.

A pose consists of its translation $t_i \in \mathbb{R}^n$ and its rotation $R_i \in SO(n)$. The group

$$SO(n) = \{R \in \mathbb{R}^{n \times n} : R^T R = I_n,\ \det(R) = 1\} \tag{2.1}$$

is the special orthogonal group for n dimensions [25, 26]. A pose corresponding to a node $\mathcal{V}_i$ will be denoted with x_i. Furthermore, $x_i = \{R_i, t_i\}$ and $x_i \in SE(n)$. All poses combined form the trajectory of the robot $x = [x_1,\ x_2, \cdots, x_n]$. The special Euclidean group $SE(n)$ is defined as follows [25]:

$$SE(n) = \left\{ \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} : R \in SO(n), t \in \mathbb{R}^n \right\}. \tag{2.2}$$

A relative pose measurement belonging to edge $\mathcal{E}_{ij} \in SE(n)$ between nodes i and j is denoted with x_{ij}. Relative pose measurements are fully described [27] by a relative rotation R_{ij} and a relative translation t_{ij} through

$$x_{ij} \left\{ \begin{aligned} t_{ij} &= R_i^T (t_j - t_i) + t_{ij}^{\epsilon} \\ R_{ij} &= R_i^T R_j R_{ij}^{\epsilon} \end{aligned} \right. . \tag{2.3}$$

Here, the matrices t_{ij}^{ϵ} and R_{ij}^{ϵ} represent the measurement noise on the translation and the rotation respectively.

With these preliminaries an optimization problem

$$\min_{\substack{R_i \in SO(n) \\ t_i \in \mathbb{R}^n}} \sum_{(i,j) \in \mathcal{E}} \mathbf{dist}_{\mathbb{R}^n}(t_{ij}, R_i^T (t_j - t_i))^2 + \mathbf{dist}_{SO(n)}(R_{ij}, R_i^T R_j)^2 \tag{2.4}$$

is stated [27, 28]. Optimization problem (2.4) formulates the aim to estimate the poses by minimizing the mismatch between the poses $x_i = \{R_i, t_i\}$ and the measurements $x_{ij} = \{R_{ij}, t_{ij}\}$, $\forall \mathcal{E}_{ij} \in G$. Details on the distance functions $\mathbf{dist}(\cdot, \cdot)$ will be given below. Optimization problem (2.4) is acquired through the maximum likelihood estimate [14, 28]. The resulting estimated poses $x^\star$ are the poses which minimize the objective function. The constraint $R_i \in SO(n)$ ensures that all matrices R_i which minimize (2.4) must be rotation matrices that adhere to (2.1). Finding a solution for (2.4) is complex because the constraint $R_i \in SO(n)$ makes the optimization problem nonconvex [24, 29].

In (2.4) the functions $\mathbf{dist}(\cdot, \cdot)$ denote various distance metrics. The function $\mathbf{dist}_{\mathbb{R}^n}(\cdot, \cdot)$ denotes the Euclidean distance

$$\mathbf{dist}_{\mathbb{R}^n}(t_{ij}, R_i^T (t_j - t_i)) = \|R_i^T (t_j - t_i) - t_{ij}\| \tag{2.5}$$

between translations, with the 2-norm denoted as $\|\cdot\|$ [27]. The aim to minimize the mismatch between the measurement on the translation t_{ij} and the relative translation between the poses i and j is clear from this equation. If minimization (2.4) was not dependent on the rotations, this minimization would simply reduce to a linear least squares problem. This highlights that the complexity lies in estimating the rotation matrices.

In this thesis, the function $\mathbf{dist}_{SO(n)}(\cdot,\cdot)$ denotes the angular distance

$$\mathbf{dist}_{SO(n)}(R_a, R_b) = \| \operatorname{Log}(R_a^T R_b)\| = \| \operatorname{Log}(R_b^T R_a)\|, \tag{2.6}$$

with $\operatorname{Log}(\cdot)$ denoting the logarithm map of a rotation matrix [27, 30]. The angular distance will be used to solve terms $\mathbf{dist}_{SO(n)}(\cdot,\cdot)$ as this is the metric which the GTSAM [18] and g^{2}o [19] libraries use. These libraries will be used to implement the algorithm proposed by this thesis. Other metrics, e.g. chordal distance or quaternion distance, have also been used to minimize (2.4) [27].

One way to minimize (2.4) is to first estimate the rotations and use the found rotations to initialize the minimization [14]. This approach is motivated by, among other reasons detailed in [27], the fact that (2.4) then reduces to two least squares problems, one of which is linear. If the rotation matrices are found in an efficient way, problem (2.4) is effectively solved. Moreover, if chordal relaxation [31] is used, problem (2.4) can be further simplified when additional approximations are applied. This is the approach of the closely related work [14]. Other approaches in finding rotation matrices to initialize optimization problem (2.4) are, among others, Riemannian gradient descent [26], semidefinite programming relaxation [32], the single loop solution [33, 34, 35] and quaternion relaxation [36] However, in this thesis, these approximations will not be applied and the nonconvex optimization problem will be solved directly. It is expected that this will produce more accurate results.

In SLAM problems, one has no prior knowledge of the map. However, to implement the theory using the GTSAM [18] and g^{2}o [19] libraries, a prior is needed. The map will not be anchored without this prior. In practice, one node is used as a prior which is set to have zero translation and zero rotation. One prior is enough to ensure observability of the global frame [6]. Using these assumptions, the set of poses that minimize (2.4) can be acquired through the nonconvex minimization

$$\begin{aligned} x^\star &= \underset{x\in SE(n)}{\mathbf{argmin}} \sum_{(i,j)\in\mathcal{E}} \left\| \operatorname{Log}\left(x_{ij}^{-1} x_i^{-1} x_j\right) \right\|^2_{\Omega_{ij}} + \sum_{i\in\mathcal{P}} \left\| \operatorname{Log}\left(\bar{x}_i^{-1} x_i\right)\right\|^2_{\Phi_{ij}} \\ &= \underset{x\in SE(n)}{\mathbf{argmin}}\ f(\mathcal{V},\mathcal{E},\mathcal{P}), \end{aligned} \tag{2.7}$$

with estimated poses $x^\star$, priors $\bar{x}_i$ and information matrices Ω_{ij} and Φ_{ij} [6]. The function $f(\mathcal{V},\mathcal{E},\mathcal{P})$ is the objective function of the PGO problem. The set of priors is denoted with $\mathcal{P}$ and has size 1 in most cases. Information matrices Ω_{ij} and Φ_{ij} are related to the measurement covariance. The notation $\|v\|^2_\Omega = v^T \Omega v$ is used. The first sum of (2.7) is equal to the sum in (2.4) through the properties of the groups $SO(n)$ and $SE(n)$ [26]. Note that (2.7) includes a prior, whereas (2.4) does not. Results of a method which uses an approximation of the problem and the results of the methods which solve the nonconvex problem directly will be compared in Section 5.1.

This concludes the discussion of the standard PGO formulation, which will be referred to as the centralized PGO formulation. Accordingly, optimization problem (2.7) is the centralized PGO problem. A distributed formulation of PGO will be discussed in the next section.

2.2 Distributed PGO: splitting formulation

While the solvers in the libraries GTSAM [18] and g^{2}o [19] acquire good results for centralized PGO problems, they do not perform well when the number of variables in the graphs grows large. The memory requirement can grow quadratically with the size of the graph, resulting in poor performance [6]. One can use iterative solvers [37, 38] to combat this issue. Here, the memory demand will only scale linearly with the number of variables. Still, the memory demand will always reach a limit [6], which means the area that can be mapped using centralized PGO is limited. One could distribute this demand over multiple units, i.e. processors or multiple robots, in order to increase the area that can be mapped using PGO. An added advantage of using multiple robots is that they can cover a larger area than one robot when given the same amount of time [5]. Furthermore, if one unit runs into a problem, e.g. a robot crashes or there is an error with one of the processors, the task will not be abandoned if the functional units take this task over [5]. This provides robustness to the system as a whole.

The distributed approach asks for a reformulation of (2.7). Traditionally, the problem is reformulated with the use of elimination [15, 16, 17]. However, methods that use elimination have to rely on linearization and are computationally expensive due to the induction of dense cliques [6]. The problem can also be reformulated with the use of separators. Using separators to reformulate the problem is beneficial because this does not require extensive bookkeeping of linearization points. Another advantage is that the sparsity of the original problem is preserved when the problem is reformulated with the use of separators. Furthermore, it has a straightforward connection to factor graphs, which means existing factor graph optimizers can be leveraged [6]. The factor graph optimizers which are included in the GTSAM [18] library can be used for this cause. Due to the reasons stated above, the graphs will be split using separators in this thesis and the problem will reformulated accordingly.

The separator method [6, 14] splits the graph into smaller subgraphs, which is illustrated in Figure 2.1. This figure shows an example of a pose graph with trajectory $[x_1, x_2, \ldots, x_9]$, which is split into two smaller subgraphs over the separating nodes x_2, x_5 and x_8. Both subgraph 1 and subgraph 2 will hold a copy of the separator. For example, subgraph 1 and subgraph 2 hold copies x_2^1 and x_2^2 of separating node x_2 respectively. The separating nodes, in this case $\{x_2, x_5, x_8\}$, will be referred to as the separators.

The separator method allows for distribution of the tasks of computation and data gathering. For example, one processor performs computations on subgraph 1 and the other processor performs computations on subgraph 2. This can be done through parallelization, which will be addressed in Chapter 3. The option to use parallelization is a major advantage of distributed PGO. The distribution of tasks is also possible in multi-robot systems because one robot can gather the measurements of subgraph 1 and another robot can gather measurements of subgraph 2. If these robots have processing units, they can perform calculations in parallel. This is not possible with the centralized PGO formulation because subgraphs cannot easily be merged.

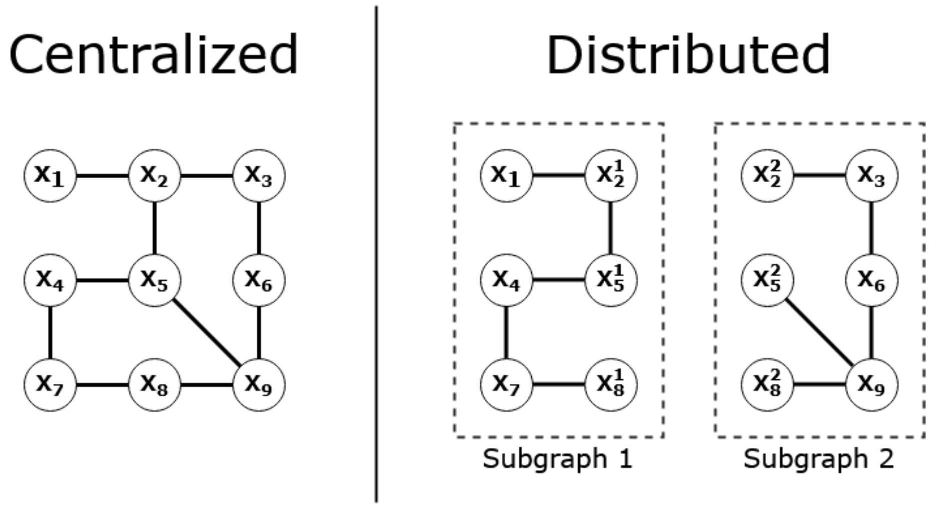


Figure 2.1: Examples of the centralized and distributed pose graphs. Poses x_i are represented as nodes in the graph, the relative pose measurements x_{ij} are represented as the edges. A copy of separator s in subgraphs g is denoted as x_g^s

After the graph is a split, a subgraph will now contain a subset of poses, measurements and priors of the original graph. The subset of poses corresponding to the nodes $\mathcal{V}_g$ of a subgraph $g \in \{1, 2, \ldots, N\} = \mathcal{G}$ will be denoted with x_g. Here, the variable N denotes the number of subgraphs. Moreover, the subset of relative pose measurements corresponding to the edges of a subgraph g will be denoted with $\mathcal{E}_g$. Lastly, the subset of priors belonging to a subgraph g is denoted as $\mathcal{P}_g$. This subset can be empty.

Each separator x^s, with $s \in \{1, 2, \ldots, S\} = \mathcal{S}$, has copies x_i^s and x_j^s in subgraph i and j respectively. The variable S denotes the number of separators. In the separator method, measurements and priors are divided over the subgraphs and therefore $\{\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_N\} = \mathcal{E}$ and $\{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_N\} = \mathcal{P}$. However, because separating nodes are copied in the respective subgraphs $\{\mathcal{V}_1 \cup \ldots \cup \mathcal{V}_N\} > \mathcal{V}$ [6]. The formulation of the problem, which is acquired through the separator method, will be referred to as the distributed PGO formulation.

Now, the optimization problem can be stated for the distributed PGO formulation as

$$\begin{aligned} \underset{x_1,\ldots,x_N \in SE(n)}{\textbf{argmin}} \quad & \sum_{g=1}^{N} f(\mathcal{V}_g, \mathcal{E}_g, \mathcal{P}_g) \\ \textbf{subject to} \quad & \mathrm{Log}((x_i^s)^{-1} x_j^s) = \mathbf{0}, \quad \forall s \in \mathcal{S}, \end{aligned} \tag{2.8}$$

with the objective function $f(\mathcal{V}_g, \mathcal{E}_g, \mathcal{P}_g)$ as in (2.7). A zero vector of appropriate size is denoted $\mathbf{0}$. Moreover, a constraint is introduced to enforce that the translation and rotation of copies x_i^s and x_j^s are identical. This constraint ensures that the estimated positions acquired through distributed PGO will be roughly equal to the estimated positions acquired through centralized PGO. This will be shown empirically in Section 5.1. Subgraphs will drift relative to each other if no constraint is added to optimization problem (2.8). Because the subgraphs are connected through their separators, all subgraphs will be anchored if there is a prior in at least one subgraph. Generally, distributed PGO problems have $N > 2$ and multi-block algorithms will be necessary to perform optimization for the problems. A multi-block formulation will be discussed in Section 3.1.2.

The advantage of the distributed formulation of PGO is that the problem can be broken into smaller subproblems. Distributed PGO is a fully separable optimization problem [39] which means distributed optimization methods, e.g. splitting methods [10], can be used. These methods allow for parallelization over different processing units. Moreover, they allow for computations on datasets which are gathered by groups of robots. The centralized formulation does not allow for the combining of maps and cannot handle such datasets. Combining maps is naturally done through the separators in the distributed PGO formulation. For the multi-robot formulation of distributed PGO the reader is referred to [14]. Apart from notation, this multi-robot formulation is equivalent to the formulation stated in this section.

One splitting method is the alternating direction method of multipliers (ADMM). It has been applied in [6] for distributed PGO. ADMM can solve separable, convex optimization problems. However, distributed PGO is a separable nonconvex optimization problem. Due to the nonconvex properties of distributed PGO, no convergence guarantees can be provided in [6]. The method in [6] is furthermore slow to converge to high accuracy. This thesis aims to expand the algorithm and introduce Nesterov acceleration [21] to speed up the convergence of ADMM in distributed PGO problems. Furthermore, the algorithm will be expanded with the globally convergent framework of [22] to ensure this convergence. This thesis will first discuss ADMM and the implementation of [6] in Chapter 3 before introducing the proposed Nesterov accelerated algorithm in Chapter 4.

Chapter 3

Alternating direction method of multipliers (ADMM) for distributed PGO

In this thesis, the alternating methods of multipliers (ADMM) [20] will be used to solve distributed pose graph optimization problems (PGO). ADMM is a widely used distributed optimization method due to its simple implementation. It can be applied in a wide range of distributed optimization problems, distributed PGO being among them [6]. It furthermore allows for parallelization, which distributes the computational load over the available units and performs the iterations in parallel for each subproblem. It allows for a practical implementation in multi-robot platforms as information exchange is minimal with ADMM [6]. Furthermore, the sparsity of the PGO problem is preserved and no approximations of the problem are necessary. It solves the nonconvex problem directly and is suspected to be more accurate for this reason.

First, a brief introduction to duality will be provided as it is the basis of ADMM. This introduction has been provided because the terminology will be used throughout this thesis. Then, in Section 3.1.1, the basic iterations of ADMM will be introduced for a standard separable optimization problem. If an optimization problem is separable the objective function consists of the sum of N functions. The ADMM iterations corresponding to an optimization problem with $N = 2$ will first be addressed in Section 3.1.1. The following section, Section 3.1.2, will address the multi-block ADMM iterations for the case with $N > 2$. Then, Section 3.2 will discuss the ADMM solution of [6] for the distributed PGO problem (2.8). The algorithm in Section 3.2 will provide the basis for the Nesterov accelerated ADMM (N-ADMM) algorithm as proposed in Chapter 4.

3.1 ADMM

In this section, the ADMM algorithm for general optimization problems will be discussed. The ADMM algorithm is a splitting scheme [10] which performs optimization on the dual formulation of an optimization problem. This dual formulation of the problem is called the dual problem. A general structure for formulating the dual problem is given in [40]. Here, a general optimization problem

$$\begin{aligned} \underset{x}{\textbf{minimize}} \quad & f_0(x) \\ \textbf{subject to} \quad & f_i(x) \leq \mathbf{0}, \quad i = 1, \ldots, m \\ & h_i(x) = \mathbf{0}, \quad i = 1, \ldots, p \end{aligned} \tag{3.1}$$

with $x \in \mathbb{R}^n$, objective function f_0, inequality constraints f_i and equality constraints h_i is stated and will be referred to as the primal problem. No assumption is made on the convexity of optimization problem (3.1) in [40].

To define the dual problem, a Lagrangian function $\mathscr{L}(\cdot)$ is stated as

$$\mathscr{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x). \tag{3.2}$$

with dual variables λ_i and ν_i. The dual problem can then be defined as

$$\begin{aligned} \underset{\lambda,\nu}{\textbf{maximize}} \quad & \inf_x \mathscr{L}(x, \lambda, \nu) \\ \textbf{subject to} \quad & \lambda \geq \mathbf{0} \end{aligned} . \tag{3.3}$$

The objective of the dual problem (3.3) is called the dual function. Functions (3.1) and (3.3) can be used to evaluate the quality of a solution to the optimization problem. To analyse the quality of a solution, the optimal value of (3.1) is denoted as $p^\star$ and the optimal value of (3.3) is denoted as $d^\star$. A value called the optimality duality gap can then be determined by $p^\star - d^\star$. Strong duality is implied when $p^\star = d^\star$, which is the case for convex problems. Otherwise, $p^\star \leq d^\star$ and weak duality holds. The duality gap has been used in the works [12, 24, 28] to evaluate the quality of solutions in PGO problems.

3.1.1 ADMM iterations

Now that the dual formulation of an optimization problem is defined, the ADMM iterations can be introduced. For the ADMM iterations a separable optimization problem

$$\begin{aligned} \underset{(x,z)}{\textbf{minimize}} \quad & f(x) + g(z) \\ \textbf{subject to} \quad & Ax + Bz = b \end{aligned} \tag{3.4}$$

is considered [20]. Here, $x \in \mathbb{R}^n, z \in \mathbb{R}^m, A \in \mathbb{R}^{p \times n}, B \in \mathbb{R}^{p \times m}$ and $b \in \mathbb{R}^p$. Furthermore, it is assumed that f and g are convex functions.

ADMM has its roots in the method of multipliers, a method which optimizes the augmented Lagrangian [20]. For optimization problem (3.4) the augmented Lagrangian becomes

$$\mathscr{L}_\rho(x,z,y) = f(x) + g(z) + y^T(Ax + Bz - b) + \frac{\rho}{2}\|Ax + Bz - b\|^2, \tag{3.5}$$

with the dual variables $y \in \mathbb{R}^p$. The Lagrangian has been augmented with the penalty factor ρ in (3.5), which is also referred to as the ρ-augmented Lagrangian.

The ADMM iterations [20] are now stated as

$$\begin{aligned} x(k+1) &:= \underset{x}{\mathbf{argmin}}\ \mathscr{L}_\rho\left(x, z(k), y(k)\right) && (3.6a)\\ z(k+1) &:= \underset{z}{\mathbf{argmin}}\ \mathscr{L}_\rho\left(x(k+1), z, y(k)\right) && (3.6b)\\ y(k+1) &:= y(k) + \rho\left(Ax(k+1) + Bz(k+1) - b\right). && (3.6c) \end{aligned}$$

Here, the iterations have initial condition $y(0) = \mathbf{0}$. First, the variables x and z are updated with the minima of the augmented Lagrangians in equations (3.6a) and (3.6b). Then, the dual variables are updated in (3.6c). Steps (3.6a) and (3.6b) are referred to as the primal updates and step (3.6c) is referred to as the dual update. These ADMM iterations can be seen as the method of multipliers where the joint minimization step is replaced with a Gauss-Seidel step [20]. The ADMM algorithm allows for separable problems because the minimization step is split in this way.

Iterations (3.6) can be rewritten in the scaled form

$$\begin{aligned} x(k+1) &:= \underset{x}{\operatorname{argmin}} \left(f(x) + \frac{\rho}{2}\left\|Ax + Bz(k) - b + u(k)\right\|^2 \right) && (3.7a)\\ z(k+1) &:= \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2}\left\|Ax(k+1) + Bz - b + u(k)\right\|^2 \right) && (3.7b)\\ u(k+1) &:= u(k) + Ax(k+1) + Bz(k+1) - b, && (3.7c) \end{aligned}$$

where $u = y/\rho$ denotes the scaled dual variable [20]. The scaled formulation will be used in this thesis and is used in the ADMM approach to distributed PGO given in [6]. This ADMM approach to distributed PGO will be discussed in Section 3.2 and will form the basis for the proposed algorithm in Chapter 4.

The connection between the ADMM approach and the closely related works of [14, 13] is clear here as well. As stated before, the algorithm can be seen as a Gauss-Seidel approach to the method of multipliers, thus connecting the ADMM approach to distributed Gauss-Seidel (DGS) method of [14]. Furthermore, the minimization steps are variants of the proximal operator [20]. For example, if (3.7a) is rewritten as

$$x(k+1) = \underset{x}{\operatorname{argmin}} \left(f(x) + \frac{\rho}{2}\left\|Ax - v\right\|^2 \right), \tag{3.8}$$

with $v = -Bz + c - u$, it is clearly visible that the righthand side is equal to the proximal operator [39] of f when $A = I$. This connects the ADMM approach to accelerated majorization-minimization (AMM) method of [13] as both methods are related to the proximal operator.

3.1.2 Multi-block ADMM

One more formulation of the ADMM algorithm is relevant to this thesis, namely the multi-block formulation. For objective functions that are separable in more than two functions the multi-block variation is needed. These kinds of objective functions take the following form [41]:

$$\begin{aligned} \min_{x_1,x_2,\dots,x_N} \quad & \sum_{i=1}^{N} f_i\left(x_i\right) \\ \textbf{subject to} \quad & \sum_{i=1}^{N} A_i x_i = b. \end{aligned} \tag{3.9}$$

Here, the functions $f_i(\cdot)$ are assumed to be convex functions. The corresponding augmented Lagrangian is given in [23] as

$$\mathcal{L}_\rho\left(x_1,\dots,x_N,y\right) = \sum_{i=1}^{N} f_i\left(x_i\right) - y^T\left(\sum_{i=1}^{N} A_i x_i - b\right) + \frac{\rho}{2}\left\|\sum_{i=1}^{N} A_i x_i - b\right\|^2. \tag{3.10}$$

Optimization problem (3.9) has been solved through variable splitting ADMM [11], Gauss-Seidel ADMM [42] or Jacobi ADMM [43]. The variable splitting ADMM approach increases the number of constraints and variables considerably for large N. This approach is considered to be inefficient [41]. Solving problem (3.9) through the Gauss-Seidel ADMM approach has proven to be more efficient, but does not allow for parallelization and may not converge for $N \geq 3$. However, empirical results have shown that it is effective for a wide range of problems [41]. The Jacobi ADMM approach to solving problem (3.9) does allow for parallelization, at the cost of being more likely to diverge [41]. For these reasons, the choice has been made to select the Gauss-Seidel ADMM approach for the proposed algorithm in Chapter 4. This is also the approach used in [6] to solve distributed PGO.

The Gauss-Seidel ADMM approach solves (3.9) through the following iterations [41]:

$$x_i(k+1) = \underset{x_i}{\textbf{argmin}}\ \mathscr{L}_\rho\left(x_1(k+1),\dots,x_i,\dots,x_N(k),y(k)\right) \quad ,\forall i \in \{1,2,\dots,N\} \tag{3.11a}$$

$$y(k+1) = y(k) - \rho\left(\sum_{i=i}^{N} A_i x_i(k+1) - b\right). \tag{3.11b}$$

Here, the primal update (3.11a) is performed sequentially for all $x_i \in \{x_1, x_2, \dots, x_N\}$ before performing the dual update (3.11b). For variable splitting ADMM and the Jacobi ADMM the reader is referred to [41]. In this report, Gauss-Seidel ADMM will simply be referred to as ADMM.

3.2 ADMM for distributed PGO

With the formulations of ADMM as described in Section 3.1, it is clear that a solution to problem (2.8) can be found with this method. Splitting schemes like ADMM [6] or the AMM scheme of [13] are suitable for solving problem (2.8) due to their simple and intuitive implementation. Furthermore, it is known that these methods reach, for many applications, an acceptable accuracy in a few iterations. However, they are also known to have slow convergence to a high accuracy solution. This can be helped by parallelization where the minimization steps of the algorithm are run simultaneously across platforms, or even across cores of the same computer. This is one of the advantages of solving the distributed problem instead of the centralized problem. Minimal information exchange between platforms is needed as it is a memory efficient approach [6]. This thesis work, along with the closely related works [6, 14, 13], have the advantage of being memory efficient and thus allows parallelization across multi-robot platforms. Other methods [15, 16, 17] rely on elimination, which is computationally expensive. Furthermore, they work with linearization points, which can introduce further problems [6]. Lastly, ADMM works directly in the nonlinear domain and it preserves the sparsity structure of the original graph [6].

For the reasons above, this thesis work uses ADMM as a solution for distributed PGO. To combat the slow convergence, the ADMM approach will be extended with Nesterov acceleration [21, 23] in Section 4.1. Note that in order to introduce parallelization to ADMM, a Jacobi approach should be considered instead of the Gauss-Seidel approach considered in this thesis [41]. Transcribing the formulations in this chapter to the Jacobi ADMM approach should not form problems, but the study of its convergence will be left for further work. Lastly, no convergence guarantees are given in [6]. This thesis aims to ensure convergence by expanding the algorithm of this section with the stabilizing framework of [22]. This expansion is given in Section 4.2. In this section, the ADMM solution to distributed PGO as introduced in [6] will be given.

First, the augmented Lagrangian, corresponding to optimization problem (2.8) is stated as

$$\mathscr{L}_\rho(x_1,\ldots,x_N) = \sum_{g=1}^{N} f\left(\mathcal{V}_g,\mathcal{E}_g,\mathcal{P}_g\right) + \sum_{s\in\mathcal{S}} (y^s)^T \operatorname{Log}\left(\left(x_i^s\right)^{-1} x_j^s\right) + \sum_{s\in\mathcal{S}} \frac{\rho}{2} \left\| \operatorname{Log}\left(\left(x_i^s\right)^{-1} x_j^s\right)\right\|^2, \tag{3.12}$$

with $y = [(y^1)^T \cdots (y^s)^T \cdots (y^S)^T]^T$ denoting the vector which stacks all Lagrange multipliers. The vector y^s denotes the dual variables that correspond specifically to separator s. Problem (2.8) is a multi-block problem (3.9) with $A_i x_i = \mathbf{0}\ \forall\ i$. If the expressions of (3.9) are substituted into the augmented Lagrangian (3.10), equation (3.12) is acquired. One notable difference between (3.10) and (3.12) is that the dual variables y^s are included in the sum in the second term of (3.12) instead of using the stacked vector as in (3.10). This is a choice for ease in implementation but constitutes the same result. Optimization problems of this form are closely related to the consensus problem as discussed in [20].

In work [6] the ADMM iterations for PGO are given as

$$x_g(k+1) = \underset{x_g\in SE(n)}{\textbf{argmin}}\ \mathscr{L}_\rho\left(x_1(k+1),\ldots,x_g,\ldots,x_N(k),y(k)\right) \qquad ,\forall g\in\mathcal{G} \tag{3.13a}$$

$$y(k+1) = y(k) + \nabla_{y(k)}\mathscr{L}_\rho\left(x_1(k+1),\ldots,x_N(k+1),y\right), \tag{3.13b}$$

with $\nabla_{y(k)}\mathscr{L}_\rho(\cdot)$ the gradient of $\mathscr{L}_\rho(\cdot)$ evaluated at $y(k)$ with respect to y. Primal updates (3.13a) are performed sequentially for every subgraph before applying the dual update (3.13b). In work [6], only two dimensions are considered and $n = 2$. Some simplification is applied to the problem in [6]. First, constants are taken out in the minimization of the Lagrangian and it is rewritten using the scaled form as in (3.7). The primal updates (3.13a) then become

$$\underset{x_i \in SE(n)}{\mathbf{argmin}}\, f(\mathcal{V}_i, \mathcal{E}_i, \mathcal{P}_i) + \sum_{s \in \mathcal{S}_i} \frac{\rho}{2} \left\| \operatorname{Log}\left((x_i^s)^{-1} x_j^s(k) \right) + \frac{y^s}{\rho} \right\|^2, \tag{3.14}$$

with $\mathcal{S}_i$ the set of separators belonging to subgraph i. In (3.14), the augmented Lagrangian is minimized. Any factor graph optimizer [18] can be used for the minimization in (3.14). Common optimizers are the Gauss-Newton method, the Levenberg–Marquardt method or the Lagrangian global optimizer which are included in the GTSAM [18] library. Lastly, the dual update (3.13b) is simplified as

$$y^s(k+1) = y^s(k) + \underbrace{\sum_{s \in x_i^s, x_j^s} \operatorname{Log}\left((x_i^s(k+1))^{-1} x_j^s(k+1) \right)}_{b^s}, \forall s \in \mathcal{S}. \tag{3.15}$$

and is performed sequentially for every separator. This formulation allows for parallel computation by splitting the dual update over the separators. One processing unit can now perform the primal update for a given subgraph and sequentially perform the dual update for the separators belonging to this subgraph. However, communication would be necessary between these units during an iteration. The underbraced term is equal to the term $\nabla_{y(k)} \mathscr{L}_\rho(\cdot)$ in (3.13b) [6] and will be denoted b^s for convenience. In the underbraced term, the sum is executed over all subgraphs that include separator x^s.

The PGO problem is inherently nonconvex [24, 29]. Moreover, no convergence proofs are available for both the centralized and distributed formulation. However, PGO has been shown to be well-behaved in terms of convergence. An analysis into this behaviour has been performed in the works [28, 44]. In this thesis, ADMM is used to perform optimization on the multi-block distributed PGO problem. Convergence of ADMM for convex problems has been studied extensively [20]. These studies on convergence were extended to the nonconvex case in, among other works, [22, 45, 46, 47]. Studies for the convex multi-block case have been done in [41, 48]. While no convergence guarantees for ADMM are available for the multi-block nonconvex case, it has shown promising results as is clear from [6] and Chapter 5.

In this chapter, an ADMM algorithm [6] was introduced which solves the distributed PGO problem. Furthermore, a theoretical connection between ADMM and the closely related methods DGS [14] and AMM [13] was shown. In the next chapter, Nesterov acceleration will be implemented in the ADMM and the necessary adaptations will be discussed. The N-ADMM algorithm that this thesis proposes will be introduced in Section 4.3.

Chapter 4

Nesterov accelerated ADMM (N-ADMM) for distributed PGO

To summarize results of Chapters 2 and 3, this thesis aims to perform optimization on the distributed pose graph optimization (PGO) problem

$$\begin{aligned} &\underset{x_1,\ldots,x_N \in SE(n)}{\textbf{argmin}} && \sum_{g=1}^{N} f(\mathcal{V}_g, \mathcal{E}_g, \mathcal{P}_g) \\ &\textbf{subject to} && \mathrm{Log}((x_i^s)^{-1} x_j^s) = 0, \quad \forall s \in \mathcal{S}, \end{aligned} \tag{4.1}$$

for which a solution was given in Chapter 3 through the alternating direction method of multipliers (ADMM). The ADMM iterations

$$x_g(k+1) = \underset{x_g \in SE(n)}{\textbf{argmin}} \, \mathscr{L}_\rho \left(x_1(k+1), \ldots, x_g, \ldots, x_N(k), y(k)\right) \qquad , \forall g \in \mathcal{G} \tag{4.2a}$$

$$y^s(k+1) = y^s(k) + b^s \qquad , \forall s \in \mathcal{S} \tag{4.2b}$$

were given in Section 3.2 as a solution for optimization problem (4.1). Here, the augmented Lagrangians are calculated through

$$\mathscr{L}_\rho(\cdot) = f(\mathcal{V}_i, \mathcal{E}_i, \mathcal{P}_i) + \sum_{s \in \mathcal{S}_i} \frac{\rho}{2} \left\| \mathrm{Log}\left((x_i^s)^{-1} \, x_j^s(k) \right) + \frac{y^s}{\rho} \right\|^2 . \tag{4.3}$$

Iterations (4.2) converge slowly and no convergence guarantees were provided in [6]. This thesis attempts to solve these issues for the ADMM approach to distributed PGO.

First, Nesterov acceleration will be implemented in Section 4.1. Then, to stabilize iterations, a stabilizing framework will be added in Section 4.2. Lastly, the proposed algorithm, which combines the choices that were made in this thesis, shall be stated in Section 4.3. The adaptations that were made to enable the implementation of acceleration and the stabilizing framework will be discussed throughout this chapter. Iterations (4.2) provide a starting point on which the building blocks of the proposed algorithm will be added.

4.1 Acceleration

Acceleration can be added in various ways [22]. For this thesis, the choice has been made for Nesterov acceleration [21] as it can acquire a provably optimal convergence rate of $\mathcal{O}(1/k^2)$ [22, 23]. Multi-block optimization methods need to be used on the distributed PGO problem because it is an optimization problem (3.9) with $N \geq 2$. Therefore, the theory of [22] cannot be used directly as it assumes $N = 2$.

In the work [23], Nesterov acceleration has been proposed for the general optimization problem (3.9) with $N \geq 2$. For performing optimization on the distributed PGO problem, the iterative scheme of [23] is rewritten and combined with the ADMM iterations (4.2) to acquire

$$\alpha(k+1) = \tfrac{1}{2} + \tfrac{1}{2}\sqrt{1 + 4\alpha(k)^2} \tag{4.4a}$$

$$x_g(k+1) = \underset{x_g \in SE(n)}{\mathbf{argmin}}\ \mathscr{L}_\rho\left(x_1(k+1), \ldots, x_g, \ldots, x_N(k), y(k)\right) \qquad ,\forall g \in \mathcal{G} \tag{4.4b}$$

$$\left.\begin{aligned} y^s(k+1) &= \hat{y}^s(k) + b^s \\ \hat{y}^s(k+1) &= y^s(k+1) + \tfrac{\alpha(k)-1}{\alpha(k+1)}\Big(y^s(k+1) - y^s(k)\Big) \end{aligned}\right\} \qquad ,\forall s \in \mathcal{S}. \tag{4.4c}$$

Algorithm (4.4) first calculates a coefficient $\alpha(k+1)$ in (4.4a), which is later used to calculate the correction term $\hat{y}^s$ for the dual update. The primal update (4.4b) is performed with the augmented Lagrangian as in (3.14). The correction term $\hat{x}_i$ of [23] can be left out in the primal update because distributed PGO is an optimization problem (3.9) with $A_i x_i = \mathbf{0}\ \forall\ i$. Nesterov acceleration is applied in the dual update (4.4c) by replacing $y^s(k)$ in (4.2) with a corrected term $\hat{y}^s(k)$, which has been calculated in the previous step. In the dual update (4.4c), $y^s(k+1)$ and $\hat{y}^s(k+1)$ are sequentially computed for every separator. Iterations (4.4) have starting conditions $\hat{y}^s(0) = \mathbf{0}$ and $\alpha(0) = 1$ [13].

When this framework is used to perform optimization on the distributed PGO problem, solutions tend to diverge. Results on this divergent behaviour are given in Figure B.1 in the appendix. To stabilize the iterations of scheme (4.4) and provide convergence, a stabilizing framework must be introduced. A stabilizing framework for ADMM is given in [22]. Here, global convergence guarantees are provided for optimization problems (3.4) with nonconvex and nonsmooth $f(x)$ and $g(z)$. This report hopes to extend this stabilizing framework to the multi-block optimization problem (3.9) as is given in the recommendations for future work in [22].

4.2 Stabilizing framework

In this section, iterations (4.4) will be extended with the stabilizing framework of [22]. To make this framework fit to the distributed PGO problem, some adaptations have been made.

These adaptations will be discussed below. The framework of [22] has been chosen as it promises convergence guarantees for nonconvex, separable optimization problems. Because distributed PGO is a nonconvex, separable optimization problem, the framework of [22] is suited for the algorithm which this thesis proposes.

Iterations (4.4) are extended with the framework as described in [22] to acquire

```
1 :  $\tau(k) = 1$
2 :  $\alpha(k+1) = \frac{1}{2} + \frac{1}{2}\sqrt{1+4\alpha(k)^2}$
3 :  $x_g(k+1) = \underset{x_g \in SE(2)}{\mathbf{argmin}}\ \mathscr{L}_\rho\left(x_1(k+1), \dots, x_g, \dots, x_N(k), y(k)\right) \quad, \forall g \in \mathcal{G}$
4 :  $\left.\begin{aligned} y^s(k+1) &= (1-\tau(k))y^s(k) + \tau(k)\hat{y}^s(k) + b^s \\ \hat{y}^s(k+1) &= y^s(k+1) + \tfrac{\alpha(k)-1}{\alpha(k+1)}\Big(y^s(k+1) - y^s(k)\Big) \end{aligned}\right\} , \forall s \in \mathcal{S}$
5 :  if $\mathscr{L}_\rho\left(x_1(k+1), \dots, x_N(k+1), y(k+1)\right) \leq \mathscr{L}_\rho\left(x_1(k), \dots, x_N(k), y(k)\right) - \sigma\|b\|^2$
     then
       $k = k+1$ go to step 1
     else
       $\tau(k) = \frac{\tau(k)}{2}$ go to step 2,
```
(4.5)

with convexity number σ and a factor $\tau(k)$, which will be discussed below. The steps of the iterations have been numbered to facilitate jumps to other steps. Starting conditions are stated in the previous sections. Furthermore, the vector $b = [(b^1)^T \cdots (b^s)^T \cdots (b^S)^T]^T$. Note that the framework of [22] was adapted to suit the multi-block distributed PGO scenario. The Lagrangians are computed accordingly. Moreover, distributed PGO is an optimization problem (3.9) with $A_i x_i = \mathbf{0}\ \forall\ i$. Consequently, the term $-\sigma\|b\|^2$ is used in the condition of step 5. The updates of [22] are replaced with the updates as in (4.4). Therefore, the dual update is performed only once in step 4. To implement the framework of [22], a factor $\tau(k)$ is introduced to balance between the plain iterations (4.2) and accelerated iterations (4.4). This factor $\tau(k)$ is discussed below. Furthermore, the penalty factor update rule of [22] updates the values for ρ and σ. However, this penalty factor update rule will not be used as this rule has destabilized the iterations in testing. Instead, the penalty factor update of [20] will be used as discussed in Section 4.3. Since no update rule for σ is present in (4.5), σ will simply be a constant.

To extend (4.4) with the framework of [22], a variable $\tau(k)$ is introduced to balance between the corrected updates $\hat{y}^s(k+1) = \hat{y}^s(k) + b^s$ if $\tau(k) = 1$ and the plain updates $y^s(k+1) = y^s(k) + b^s$ if $\tau(k) = 0$. In other words, for $\tau(k) = 1$ steps 2, 3 and 4 will be equal to iterations (4.4) and for $\tau(k) = 0$ steps 2, 3 and 4 will be equal to iterations (4.1). If the Nesterov accelerated iterations do not cause a decrease on the augmented Lagrangian larger than $\sigma\|b\|^2$, variable $\tau(k) \to 0$ and the algorithm will revert to the plain ADMM iterations.

If the condition in step 5 of (4.5) is not met, $\tau(k)$ is decreased and the algorithm goes back to step 3. However, at certain iterations the condition in step 5 is never met even as $\tau(k) \to 0$. Hence, in this form the rule of step 5 can cause the algorithm to get caught in an infinite loop which never converges to a solution. To avoid this infinite loop, the rule of step 5 has been extended with a limit on the number of times this condition is not met per iteration. This adaptation is discussed in Section 4.3.

4.3 Proposed algorithm

In order to make the framework of [22] suitable for solving distributed PGO problems, the penalty factor update rule of [22] will not be included in the proposed algorithm. Instead, the penalty factor update rule of [20] will be implemented. In testing, the penalty factor update rule of [22] has shown to destabilize solutions as the penalty factor will tend towards infinity as the iteration number increases. The penalty factor update rule of [20] is

$$\rho(k+1) = \begin{cases} \gamma\rho(k) & \text{if } p_{\text{res}} > \mu d_{\text{res}} \\ \frac{1}{\gamma}\rho(k) & \text{if } d_{\text{res}} > \mu p_{\text{res}} \\ \rho(k) & \text{otherwise,} \end{cases} \tag{4.6}$$

with p_{res} and d_{res} the primal and dual residual respectively. The method to determine the value of the residuals will be discussed below and is calculated through (4.8). Parameters $\gamma > 1$ and $\mu > 1$ can be chosen as desired.

Based on the choices and considerations made in this thesis, the following Nesterov accelerated ADMM (N-ADMM) algorithm is proposed:

```
1 : if $p_{res} > 10d_{res}$
    then
      $\rho(k+1) = 2\rho(k)$
    if $d_{res} > 10p_{res}$
    then
      $\rho(k+1) = \frac{\rho(k)}{2}$
2 : $\tau(k) = 1, m = 0$
3 : $\alpha(k+1) = \frac{1}{2} + \frac{1}{2}\sqrt{1+4\alpha(k)^2}$
4 : $x_g(k+1) = \underset{x_g \in SE(n)}{\textbf{argmin}}\ \mathscr{L}_{\rho(k+1)}(x_1(k+1), \dots, x_g, \dots, x_N(k), y(k)) \quad , \forall g \in \mathcal{G}$
5 : $\left.\begin{aligned} y^s(k+1) &= (1-\tau(k))y^s(k) + \tau(k)\hat{y}^s(k) + b^s \\ \hat{y}^s(k+1) &= y^s(k+1) + \frac{\alpha(k)-1}{\alpha(k+1)}\Big(y^s(k+1) - y^s(k)\Big) \end{aligned}\right\}, \forall s \in \mathcal{S}$
6 : if $\mathscr{L}_{\rho(k+1)}(x_1(k+1), \dots, x_N(k+1), y(k+1)) \leq \mathscr{L}_{\rho(k+1)}(x_1(k), \dots, x_N(k), y(k)) - \sigma\|b\|^2$
    or $m = M$
    then
      $k = k+1$ go to step 1
    else
      $\tau(k) = \frac{\tau(k)}{2}$
      $m = m+1$ go to step 3.
```
(4.7)

As stated before, algorithm (4.7) has starting conditions $y^s(0) = \hat{y}^s(0) = \mathbf{0}$ and $\alpha(0) = 1$. The penalty factor update rule (4.6) with $\gamma = 2$ and $\mu = 10$ is implemented in step 1 of (4.7). These values were shown to be effective during testing.

In work [22], a varying $\sigma(k)$ dependent on $\rho(k)$ is proposed. However, since the penalty factor update rule of [20] is used instead, $\sigma = 1$ is chosen. This was empirically proven to be effective. Starting condition $\rho(0)$ can be chosen freely. For large $\rho(0)$ however, the iterations will not converge. Certain starting values $\rho(0)$ cause the iterations to converge faster. Finding values $\rho(0)$ that perform well is not trivial: a certain value can perform well for certain datasets but poor for other datasets.

To avoid the infinite loop as described in Section 4.2, the variable m is introduced. This variable counts the number of times the condition on the Lagrangians in step 6 of (4.7) is not met. If m is equal to the maximum M, the algorithm moves to the next iteration regardless of the condition on the Lagrangians.

Algorithm (4.7) uses the stopping conditions as defined in [6]. These stopping conditions are met when the values of the primal and dual residuals reach a certain threshold. The threshold can be set as desired according to the desired accuracy. The primal and dual residuals are determined by

$$p_{res} = \sum_{s \in \mathcal{S}} \left\| \mathrm{Log}\left(\left(x_i^{s,\star}\right)^{-1} x_j^{s,\star} \right) \right\| \tag{4.8a}$$

$$d_{res} = \left\| \nabla_x \mathcal{L}\left(x_1^\star, \dots, x_N^\star, y\right) \right\|. \tag{4.8b}$$

The gradient $\nabla_x \mathcal{L}(\cdot)$ in (4.8b) is calculated according to the unaugmented Lagrangian, which is equal to the sum of the first two right-hand side terms in (3.12). Optimality is achieved when $\mathrm{Log}((x_i^{s,\star})^{-1} x_j^{s,\star}) = \mathbf{0}\ \forall s \in \mathcal{S}$ and $\nabla_x \mathcal{L}\left(x_1^\star, \dots, x_N^\star, y\right) = \mathbf{0}$ [6].

In this chapter, the N-ADMM algorithm was introduced. It was constructed by combining, adapting and extending the current literature [6, 20, 22, 23]. The choices made were widely discussed in this chapter. The performance of N-ADMM will be compared with the state of the art in the next chapter.

Chapter 5

Performance of N-ADMM

Based on the considerations made throughout this work, the alternating direction method of multipliers (ADMM) for distributed pose graph optimization (PGO) was extended with Nesterov acceleration to create the Nesterov accelerated ADMM (N-ADMM) algorithm (4.7). The proposed N-ADMM algorithm has been implemented in C++ to perform optimization on the distributed PGO problem for various datasets. The code can be found through the following links: https://cutt.ly/6j5IbYS and https://drive.google.com/drive/folders/1pYNxD_RHwOPQEJOXdn6pfiK5VmAvLxYc. In this chapter, the results will be documented.

Datasets are in .g2o format [19] and consist of the nodes and edges corresponding to a pose graph. The edges correspond to the measurements collected by the robot and the nodes correspond to positions based on an initial guess from these measurements. Datasets may be created in simulation and always come with these initial guesses. In this thesis, only two-dimensional scenarios are considered.

To implement N-ADMM, the Gauss-Newton method is used to perform the minimization in the primal updates. This method is provided in the GTSAM library. All other steps of (4.7) are programmed using the classes, structures and functions of the GTSAM library. Some functionalities of the publicly available code of [6] has been used. Before N-ADMM can be applied, the graphs provided in the datasets have to be partitioned into subgraphs. This is necessary because only datasets that were collected by a single robot are publicly available. To solve this issue, the METIS algorithm [49] is used to partition a graph into N subgraphs. This effectively simulates datasets that have been collected by multiple robots because the multi-robot formulation [14] is equivalent to the formulation in Section 3.2. The publicly available code of [6] is used for this partitioning. The datasets that are created by the models in Section 5.2 are also partitioned in this way for consistency.

First, N-ADMM will be compared with plain ADMM [6] and the state of the art [15, 16, 17, 14, 13] in Section 5.1. Within the state of the art, the distributed Gauss-Seidel (DGS) [14] and accelerated majorization-minimization (AMM) [13] methods are considered to be the closest related works due to the theoretical connection as discussed in Sections 2.2 and 3.1.1. From the tests on the benchmark datasets, it is not clear what characterizes the datasets for which N-ADMM is most efficient. For this reason, two models will be introduced in Section 5.2 to investigate the effect of several characteristics of the datasets on efficiency.

Table 5.1: Number of nodes and edges in the publicly available datasets

Dataset	Nodes	Edges
FR079	989	1217
CSAIL	1045	1172
Intel	1728	2512
M3500	3500	5598
AIS2klinik	15115	16727

5.1 Comparison with state of the art

In this section, the performance of N-ADMM will be compared with the state of the art methods for distributed PGO by performing tests on the publicly available datasets. Performance was judged based on efficiency and accuracy. Efficiency is measured by the number of iterations needed and the time needed until convergence. The value of the objective function $f(\cdot)$ at convergence is a good indication of the accuracy of the compared methods.

To compare the performance of N-ADMM to ADMM [6], tests were run on the publicly available datasets FR079, CSAIL, Intel, M3500 [50] and AIS2klinik [17]. Together, they provide a complete mix of scenarios. Datasets FR079, CSAIL and Intel have been obtained by processing measurements of a robot equipped with a laser range finder. The pose graphs have been created by combining data from the wheel odometry and data from the laser range finder. The datasets FR079, CSAIL and Intel have been acquired in indoor office environments. The dataset AIS2klinik is collected by a similar robot, but is acquired in an outdoor setting. It is furthermore the largest dataset. The dataset M3500 models a robot driving on a grid and has been acquired through simulation. Table 5.1 details the number of nodes and edges in the pose graphs and it is clear that a wide range of graph sizes is available with the benchmark datasets.

The code for the plain ADMM iterations of [6] is publicly available. This thesis extends the code of [6] with the steps as in (4.7) to acquire the code for the N-ADMM algorithm. To compare the relative efficiency of N-ADMM to ADMM, the algorithms were run sequentially on the same machine until the stopping condition on the residuals was reached. These tests were run for $\rho(0) = 0.2$ and $M = 3$ on the publicly available datasets and the results are gathered in Table 5.2. Furthermore, the graph has been partitioned into ten subgraphs, simulating the scenario for a system with ten robots. As a measure of efficiency of N-ADMM and ADMM, the number of iterations that were needed to reach both stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$ were noted for both algorithms. Furthermore, the time needed and the value of the objective function $f(\cdot)$ were recorded when the stopping condition was reached.

For the dataset AIS2klinik, a significant reduction in the number of iterations is acquired. In this case, this reduction translates into a large reduction in time needed. Because iterations are more time consuming for larger graphs, costly computations are avoided with the N-ADMM algorithm. However, for the dataset M3500, the reduction in the number of iterations is not translated into a reduction in the time needed for convergence. This is due to the fact that the individual iterations of the N-ADMM algorithm take longer than the iterations of the

Table 5.2: Comparison of results between the N-ADMM and ADMM algorithms for the publicly available datasets. Results were acquired with $M = 3$, $N = 10$, $\rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$.

Dataset	Iterations needed		Time needed [s]		$f(\cdot)$	
	N-ADMM	ADMM	N-ADMM	ADMM	N-ADMM	ADMM
FR079	4	4	0.50	0.46	0.09	0.08
CSAIL	11	8	1.15	0.83	0.39	0.43
Intel	245	245	47.74	32.78	45.07	45.07
M3500	112	148	52.05	51.48	148.45	148.15
AIS2klinik	115	197	182.03	230.86	174.37	174.42

Table 5.3: Further time reduction by varying M. Results are only stated for values of M which reduced the time needed most. Acquired with $N = 10$, $\rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$. This table has been visualized in the appendix in Figure B.2

Dataset	Optimal M	Iterations needed		Time needed [s]		$f(\cdot)$	
		N-ADMM	ADMM	N-ADMM	ADMM	N-ADMM	ADMM
Intel	1	245	245	36.97	32.03	45.07	45.07
M3500	4	100	148	49.49	51.71	148.59	148.15
AIS2klinik	1	101	197	140.91	230.25	174.47	175.42

plain ADMM algorithm. For this same reason, the N-ADMM algorithm takes longer for the datasets Intel, CSAIL and FR079, where equal or more iterations are needed. Furthermore, it is clear that both methods converge to solutions with the same quality because the difference in the value for the $f(\cdot)$ is negligible.

These results indicate that the N-ADMM algorithm is most efficient for larger datasets as a large reduction in the number of iterations is acquired for the datasets AIS2klinik and M3500. Furthermore, in an absolute sense, the time needed can be reduced most for large datasets, which means the N-ADMM algorithm might be very useful in these cases. To investigate if there is a relation between graph size and the relative efficiency of the N-ADMM algorithm, new datasets must be created in simulation because the datasets in Table 5.1 are the only publicly available datasets for the PGO problem. In Section 5.2, two models will be introduced to create new datasets. With these models, various parameters can be varied to investigate what characterizes the datasets for which N-ADMM is most efficient. Below, this thesis will only consider the largest datasets Intel, M3500 and AIS2klinik.

To investigate the effect of M on the performance of the N-ADMM algorithm, tests were run for various values of M. The results are stated in Table 5.3 for the values of M which produce the best outcomes time-wise. A visualization is available in Figure B.2 and the full results are available in Table B.1 in the appendix. For the dataset AIS2klinik, the difference in iterations and time needed is even more drastic than in Table 5.2. Furthermore, N-ADMM now needs less time for convergence than ADMM for the dataset M3500. Moreover, the difference in time is reduced for the dataset Intel. It is also clear that picking a good value for M is not trivial as different values are optimal for different datasets.

To compare N-ADMM with the state of the art, this thesis has to rely on the results from [6, 13] as the code for the algorithms are not always publicly available and implementation

Table 5.4: Value of $f(\cdot)$ at $k = \{100, 250, 1000\}$. Results achieved with $M = 3, N = 10$ and $\rho(0) = 0.2$. The results of the methods marked with an asterisk "*" are retrieved from [13]. This table is visualized in Figure B.3

Dataset	Maximum iterations	$f(\cdot)$			
		N-ADMM	ADMM [6]	AMM [13]*	DGS [14]*
Intel	100	45.43	45.43	52.52	52.55
	250	45.07	45.07	52.48	52.44
	1000	45.01	45.01	52.40	52.38
M3500	100	148.6	148.8	194.7	195.6
	250	147.6	147.6	194.4	194.6
	1000	146.8	146.9	194.0	194.3
AIS2klinik	100	174.9	183.5	198.2	864.6
	250	174.2	174.2	196.2	931.5
	1000	174.0	173.9	193.3	335.0

is often complex if they are. First, N-ADMM will be compared with the closest related methods DGS [14] and AMM [13]. To make this comparison, optimization was performed with the methods N-ADMM and ADMM under the same conditions as in [13]. In that work, optimization is performed until a maximum number of iterations has been reached, which takes the place of the stopping conditions as in Section 4.3. The results are recorded in Table 5.4. For easier comparison, these results are visualized in Figure B.3 of the appendix.

When the results in Table 5.4 are compared with the results in [13], it can be seen that both N-ADMM and ADMM outperform AMM and DGS significantly and a solution with superior quality is acquired in fewer iterations. Furthermore, N-ADMM reaches a better solution in fewer iterations, but might not reach the best solution for a higher maximum number of iterations. However, the difference in the values for $f(\cdot)$ for N-ADMM and ADMM are already negligible at 250 iterations. The DGS method approximates the distributed PGO problem, whereas N-ADMM, ADMM and AMM solve the nonlinear optimization problem directly. In this test, there is a clear gap in accuracy between the method that uses approximations of the distributed PGO problem and the methods that solve the distributed PGO problem directly.

To compare the performance of N-ADMM and ADMM with the tectonic smoothing and mapping method (TSAM2) [15], the hierarchical optimization on manifolds method (HOGMan) [16] and the small-memory footprint method (SMF) [17], tests were run for N-ADMM and ADMM under the same conditions as in [6]. In the work [6], optimization was terminated when either the maximum number of iterations was reached or both stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$ were acquired. The results are summarized in Table 5.5. In this table, the time needed is not considered as the tests have been run on a different machine than the machine that was used in [13]. Therefore, results regarding computation times cannot be compared. The results of Table 5.5 are visualized in Figure B.4 of the appendix for easier comparison.

When results are compared between [6] and the results achieved by N-ADMM and ADMM, it is clear that only TSAM2 outperforms N-ADMM and ADMM when it comes to accuracy. However, when compared to the results of the other methods in [6], the difference between N-ADMM, ADMM and TSAM2 is negligible. It is important to note that TSAM2, HOG-Man and SMF will most likely be faster time-wise than N-ADMM and ADMM. Surprisingly,

Table 5.5: Results for a maximum of 200 iterations with $M = 3,\ N = 10,\ \rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$. The results of the methods marked with an asterisk "*" are retrieved from [6]. This table is visualized in Figure B.4 in the appendix

Dataset	$f(\cdot)$				
	N-ADMM	ADMM [6]	TSAM2 [15]*	HOGMan [16]*	SMF [17]*
Intel	45.11	45.11	45.0	134.7	53.3
M3500	148.45	148.15	146.1	521.9	287.1
AIS2klinik	174.37	174.42	172.8	647.0	471.0

ADMM performs better in this thesis than in [6]. Both N-ADMM and ADMM acquire values for $f(\cdot)$ close to those achieved through the centralized formulation [6].

To summarize results, N-ADMM converges in fewer iterations than ADMM in some cases. Because the individual iterations of N-ADMM take longer than those of ADMM, this reduction in iterations is not always translated into a reduction in the time needed for convergence. If parameters are selected to be optimal, a significant reduction can be acquired in the time needed for convergence. In the best case, N-ADMM is 40% more efficient time-wise than ADMM. However, for most datasets, N-ADMM is roughly as efficient time-wise as ADMM. Both N-ADMM and ADMM acquire similar values for $f(\cdot)$ at convergence, with negligible difference. This verifies that the solutions achieved by N-ADMM are of equal quality as those achieved by ADMM. When the accuracy of N-ADMM and ADMM is compared to the accuracy of the closest related works [14, 13], it is clear that they achieve solutions of superior quality in less iterations. Compared to the methods [15, 16, 17], only [15] performs slightly better.

N-ADMM performs better than ADMM for some datasets. However, it is unclear what characterizes the datasets for which N-ADMM is most efficient. To investigate this, two models will be formulated in Section 5.2 and datasets will be created based on these models. Several parameters can be varied in these models such that correlations can be found between the efficiency of N-ADMM and these parameters. The choice was made in this thesis to focus on graph size and bad initial guesses because these were thought to be of the most influence on dataset complexity.

5.2 Generated datasets

In this section, two models will be introduced which simulate a robot trajectory. These models were created such that the parameters of interest could be adjusted. Datasets were created with these models using the GTSAM [18] and g^{2}o [19] libraries and the classes and formats they provide. By doing so, insight can be acquired into the relationship between the characteristics of the datasets and the efficiency of N-ADMM. First, the model which will be referred to as the block model will be used to investigate the effect of graph size. Then, a model which will be referred to as the Gaussian model will be introduced to investigate the effects of varying the variance and mean of the offsets on the relative pose measurements. In this way, the effect of bad initial guesses will be tested. Graph size will also be investigated with the Gaussian model.

Both models simulate a robot moving in a linear trajectory. An initial guess for pose x_j, with $j \in \{1, 2, \dots, J-1\}$ and J the number of poses in the generated graph, is generated through

$$x_j = \begin{bmatrix} x_j^x \\ x_j^y \\ x_j^\theta \end{bmatrix} = \begin{bmatrix} j + \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix}. \tag{5.1}$$

Here, the variables $\tilde{x}, \tilde{y}$ denote the offsets in x and y directions. The variable $\tilde{\theta}$ denotes the offset on the angle. Furthermore, $x_0 = \mathbf{0}$. This creates a linear trajectory in x-direction with slight offsets. A relative pose measurement is generated through

$$x_{ij} = \begin{bmatrix} x_{ij}^x \\ x_{ij}^y \\ x_{ij}^\theta \end{bmatrix} = \begin{bmatrix} x_j^x - x_i^x + \hat{x} \\ x_j^y - x_i^y + \hat{y} \\ x_j^\theta - x_i^\theta + \hat{\theta} \end{bmatrix}, \tag{5.2}$$

with $i = j-1$. The variables $\hat{x}, \hat{y}$ and $\hat{\theta}$ denote the offsets on the measurements. To create a factor in the pose graph when using GTSAM, an information matrix Ω_{ij} is necessary. For the purposes of this thesis, the information matrix belonging to the relative pose measurements of dataset M3500 is used. This dataset is a simulated dataset, with properties similar to the datasets which the models of this section generate. Through these means a trajectory can be created with J poses and $J-1$ relative pose measurements. The offsets are calculated differently for each model and these will be introduced first for the block model.

5.2.1 Block model

For the block model, the offsets are calculated through

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix} = \begin{bmatrix} \pm\, 0.11 \left(\frac{\mathcal{U}(0,1000)}{16000}\right) \\ \pm\, 0.11^2 \left(\frac{\mathcal{U}(0,1000)}{16000}\right) \\ \pm\, 0.11 \left(\frac{\mathcal{U}(0,1000)}{16000}\right) \end{bmatrix}, \quad \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} \pm\, \left(0.015 + 0.5 \left(\frac{\mathcal{U}(0,1000)}{16000}\right)\right) \\ \pm\, \left(0.015 + 0.5 \left(\frac{\mathcal{U}(0,1000)}{16000}\right)\right) \left(0.0015 + \left(\frac{\mathcal{U}(0,1000)}{16000}\right)\right) \\ \pm\, \left(0.0015 + \left(\frac{\mathcal{U}(0,1000)}{16000}\right)\right) \end{bmatrix}. \tag{5.3}$$

Here, $\mathcal{U}(a,b)$ denotes the uniform distribution on the open interval (a,b). For the block model, $\mathcal{U}(a,b)$ is implemented with the `C++` function `rand()`. The operator $\pm$ denotes that there are equal chances of the sign being positive or negative. This sign is determined using `rand()` as well. A sketch of the model can be found in Figure 5.1. There is a high degree of randomness to this model because a measured pose can fall into any of the four boxes. Difficulty in optimization could stem from this randomness. It is not a very realistic model for a robot as sensors do not tend to create these kind of patterns in the relative pose measurements. However, it has been made with the intent to create datasets which are difficult to perform PGO on. This will become apparent below.

With this model, datasets of varying graph size were created in the `.g2o` format. Then, the graphs are partitioned into 25 subgraphs and optimization is performed by the N-ADMM

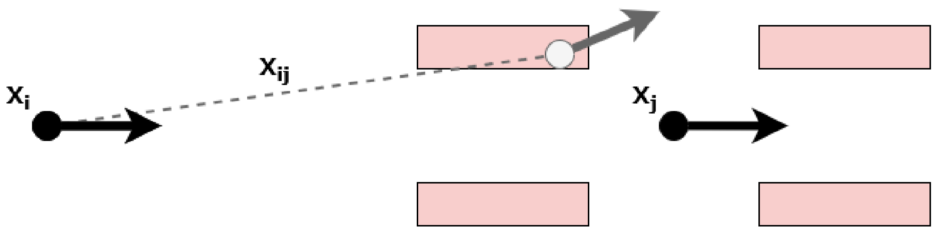
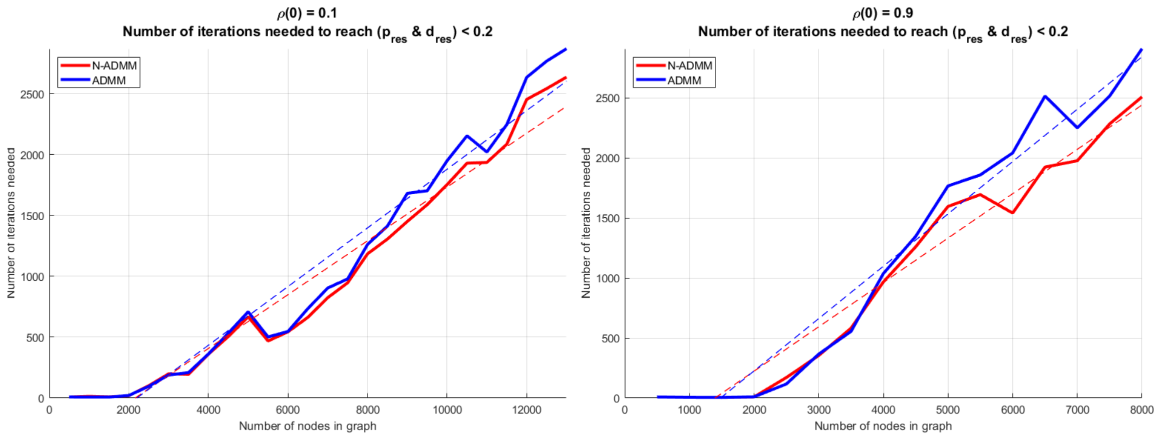


Figure 5.1: Sketch of the block model. Initial guesses are shown with the black arrows x_i and x_j. The relative pose measurement is displayed as the grey dotted line x_{ij} and the grey arrow indicates the fictional measured pose that corresponds to the pose measurement. The red blocks indicate the possible locations of these fictional measured poses. These blocks are determined by (5.3) and are not to scale in this sketch.



(a) Results for $\rho(0) = 0.1$ **(b)** Results for $\rho(0) = 0.9$

Figure 5.2: Number of iterations needed to reach $p_{res} < 0.2$ and $d_{res} < 0.2$ for a given number of nodes J. Performance of N-ADMM is given in red, performance of ADMM is given in blue. Here, $M = 3$ and $N = 25$.

and ADMM methods for $\rho(0) = 0.1$ and $\rho(0) = 0.9$ until both stopping conditions $p_{res} < 0.2$ and $d_{res} < 0.2$ are reached. Stopping conditions were relaxed because optimization on these datasets proved to be difficult. Optimization was performed for increasing graph sizes until one method reached 3000 iterations. The results are given in Figure 5.2. Here, it can be seen that the N-ADMM method needs less iterations to converge than the ADMM methods when the graph size exceeds 4000 nodes. However, since the N-ADMM needs roughly 10% less iterations, this reduction in the number iterations does not translate into a reduction in the time needed for convergence. For both methods, similar values for $f(\cdot)$ were achieved.

It is clear that the block model produces graphs that are difficult to perform optimization on. For example, the dataset AIS2klinik has 15115 nodes in its graph, but both N-ADMM and ADMM need less than 200 iterations to converge to stricter stopping conditions. It can also be seen that the difficulty scales with the number of poses in the graph. It is unclear from the graph itself what causes this difficulty. For this reason, the Gaussian model is introduced in the next section to investigate the effect of the mean and variance of the relative pose measurements.

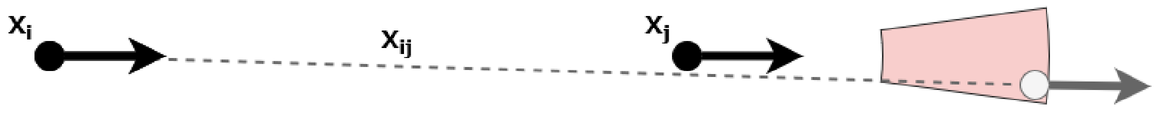


Figure 5.3: Sketch of the Gaussian model. The notation as in Figure 5.1 is used. Not to scale.

5.2.2 Gaussian model

For the Gaussian model, the offsets are calculated through

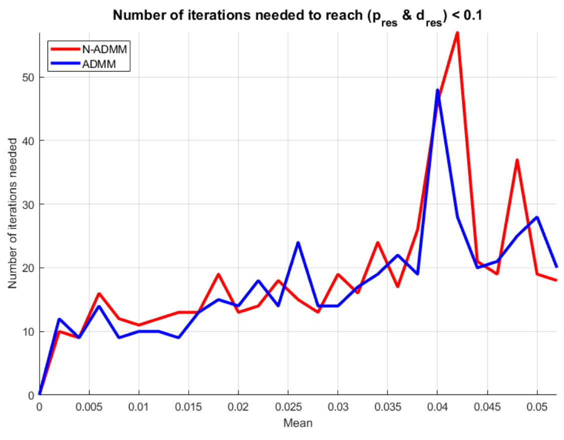
$$\begin{bmatrix}\tilde{x}\\ \tilde{y}\\ \tilde{\theta}\end{bmatrix} = \begin{bmatrix}\mathcal{N}(0,0.05)\\ \mathcal{N}(0,0.05)\\ 0\end{bmatrix}, \quad \begin{bmatrix}\hat{x}\\ \hat{y}\\ \hat{\theta}\end{bmatrix} = \begin{bmatrix}\cos\left(\mathcal{U}\left(-1{\times}10^{-6}, 1{\times}10^{-6}\right)\right)\mathcal{N}\left(\mu,\sigma\right)\\ \sin\left(\mathcal{U}\left(-1{\times}10^{-6}, 1{\times}10^{-6}\right)\right)\mathcal{N}\left(\mu,\sigma\right)\\ 0\end{bmatrix}, \tag{5.4}$$

with $\mathcal{N}(c,d)$ denoting the Gaussian distribution with mean c and standard deviation d. Here, the random generator `mt19937` is used in `C++` to draw from the distributions $\mathcal{N}(\cdot)$ and $\mathcal{U}(\cdot)$. A sketch of the Gaussian model is found in Figure 5.3. This model is considered to be a more realistic representation of a robot as offsets on the relative pose measurements are connected to the orientation of the robot. Furthermore, sensors can be modelled better with Gaussian distributions than uniform distributions.

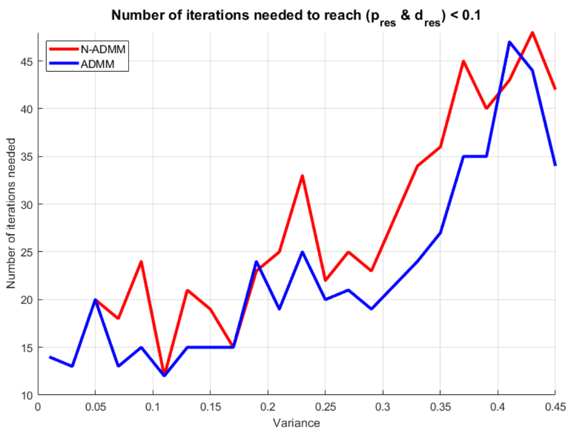
Datasets were created with this model for varying μ and σ. Then, optimization was performed with the N-ADMM and ADMM methods with $\rho(0) = 0.1$ and $\rho(0) = 0.9$. The datasets were partitioned into 25 subgraphs. Stopping conditions were set to $p_{res} < 0.1$ and $d_{res} < 0.1$ because optimization proved to be easier on these datasets. The results of these tests are given in Figure 5.4. Optimization was performed for increasing values of μ and σ until one method could no longer converge. In these tests, increasing μ and σ led to an increase in the number of iterations needed for convergence. N-ADMM and ADMM show similar performance, but ADMM outperforms N-ADMM slightly when the variance is increased. Both methods achieve similar values for $f(\cdot)$. No correlations could be found when different values for $\tilde{x}, \tilde{y}, \tilde{\theta}$ and $\hat{\theta}$ were given.

However, when datasets were created for increasing graph sizes, with fixed values for μ and σ, a correlation as in Figure 5.1 could not be perceived. Moreover, roughly the same number of iterations were needed for convergence for datasets with graph sizes ranging from 2000 nodes to 25000 nodes. Approximately 40 iterations were needed by both methods on this range. An explanation for this behaviour could be that there is a certain complexity per pose in the block model which is not present in the Gaussian model. What causes this complexity is unknown. Since the correlation between graph size and the efficiency of N-ADMM is not present in the Gaussian model, it cannot be concluded that there is a relationship between the relative efficiency of N-ADMM and large graph sizes alone.

To conclude, while it is clear from Section 5.2.1 that efficiency can be increased significantly when using N-ADMM, it is not clear what characterizes the datasets for which N-ADMM is more efficient. More research is necessary to find this correlation. For further research, the effect of the information matrix could be examined. Furthermore, a linear trajectory without loop closures is taken in this thesis. Future research could examine the effect of loop



(a) Results for $\sigma = 1{\times}10^{-6}$ and varying μ



(b) Results for $\mu = 2{\times}10^{-2}$ and varying σ

Figure 5.4: Number of iterations needed to reach $p_{res} < 0.1$ and $d_{res} < 0.1$ for varying values μ and σ. Performance of N-ADMM is given in red. Performance of ADMM is given in blue. Here, optimization was performed with $M = 3$ and $N = 25$ on a graph with 10000 nodes. Datasets were created with the Gaussian model.

closures and more elaborate trajectories. Moreover, N-ADMM was only tested in simulation. Collecting the data in experimentation could be a further path of investigation. Lastly, the inclusion of outliers in the datasets are known to complicate computations and their effect could be studied. The choice was made to not further investigate these paths and focus on the creation and implementation of the N-ADMM algorithm. Further investigation would require extensive research into pose graph complexity and simulation. Moreover, it would be too time-consuming to set up experiments and it would require resources that are not currently available. Even if these paths were researched, it would not guarantee that the exact scenarios in which N-ADMM is more efficient than ADMM can be found.

Lastly, the solutions of N-ADMM and ADMM are very accurate and they provide a good option when accuracy is important. As is clear from Table 5.4, N-ADMM converges faster to a lower value for $f(\cdot)$ than ADMM. Moreover, this thesis has shown that both N-ADMM and ADMM reach a lower value for $f(\cdot)$ much faster than AMM and DGS. However, N-ADMM and ADMM are likely to be time-consuming options. This might be nullified by introducing parallelization as discussed in Chapter 3.

Chapter 6

Conclusions

In this thesis, a new, accelerated algorithm was created for distributed pose graph optimization (PGO) problems. Acceleration was implemented in the alternating direction method of multipliers (ADMM) to create Nesterov accelerated ADMM (N-ADMM) (4.7). The N-ADMM algorithm was tested on the publicly available benchmark datasets to compare its performance to the state of the art. Two new models were created to gain insight into the effect of graph sizes and bad initial guesses on the performance of N-ADMM. In all tests, the methods N-ADMM and ADMM were run sequentially to investigate the relative effect of acceleration.

6.1 Answers to the research questions

The following main research question was stated:

"How can Nesterov acceleration be implemented in the ADMM algorithm for solving distributed PGO problems and what adaptations have to be made to the current methods for this implementation?"

To implement Nesterov acceleration in ADMM for distributed PGO, the multi-block formulation of Nesterov acceleration [23] was used to extend the algorithm of [6]. By doing so, algorithm (4.4) was acquired. However, this algorithm does not converge in testing. The stabilizing framework of [22] was implemented to obtain (4.5). However, adaptations were necessary to create a convergent algorithm. The necessary adaptations were widely discussed in Section 4.2. Adaptations to the current methods were made based on the existing literature and empirical results. Because the penalty factor update rule of [22] caused divergence, the penalty factor of [20] was used to create the N-ADMM algorithm. Moreover, a rule was introduced in this thesis to avoid infinite loops. Through combining, extending and adapting the current methods to solve distributed PGO problems, the N-ADMM algorithm (4.7) was created.

Based on the choices made throughout this work, a method which implements Nesterov acceleration in ADMM for distributed PGO was introduced in this thesis. However, other choices can be made which might lead to a different method that implements Nesterov acceleration in ADMM for distributed PGO. A few examples for further research to consider will be given in Section 6.2. Methods based on other choices might perform better than N-ADMM in some aspects, but worse in other aspects. For example, if Jacobi updates are used for the ADMM iterations as discussed in [41], parallelization can be applied in computing. This would allow for a drastic reduction in the time needed for convergence. However, this may come at the cost of a higher likelihood of divergence.

To judge the value of N-ADMM in the context of the state of the art, the following subquestion was raised:

"How does the performance of N-ADMM compare to the state of the art?"

In Section 5.1, a comparison was made between N-ADMM and the state of the art. Performance was measured in terms of efficiency and accuracy. Efficiency was determined based on the number of iterations needed for convergence and the time needed for convergence. Accuracy was measured by the value of the objective function at convergence. The works [6, 14, 13] were considered to be the closest related works as they formulate distributed PGO with the use of separators as discussed in Section 2.2. Furthermore, the N-ADMM algorithm is an extension of the ADMM approach of [6]. Because N-ADMM uses Gauss-Seidel steps, the distributed Gauss-Seidel (DGS) method [14] also has connections to N-ADMM. Moreover, the minimization steps of N-ADMM are variants of the proximal operator, which is also used in the accelerated majorization-minimization (AMM) method [13]. To compare N-ADMM and ADMM, both methods were run sequentially on the same datasets. This was possible due to the open source code of ADMM for distributed PGO [6]. For comparison with the DGS method [14] and the AMM method [13], this thesis had to rely on the results that were stated in [13].

When compared to ADMM, N-ADMM has shown to have better performance because a more accurate solution was found in fewer iterations. In some cases, this can translate into a significant reduction in the time needed when N-ADMM is used. Under optimal conditions, this reduction in the time needed becomes more distinct. Moreover, in the cases where N-ADMM is outperformed by ADMM, the difference in the time needed becomes negligible when optimal parameters are selected. The difference in accuracy when N-ADMM and ADMM are converged is negligible. However, as stated before, N-ADMM will reach this accuracy in fewer iterations. Both N-ADMM and ADMM outperform the methods of DGS and AMM significantly and they reach more accurate solutions much faster. Moreover, a gap can be seen between DGS, a method which uses approximations for the formulation of distributed PGO, and N-ADMM, ADMM and AMM. The methods N-ADMM, ADMM and AMM use no approximations in formulating and solving the distributed PGO problem and these results indicate that these are more accurate for this reason.

When the performance of N-ADMM is compared to the performance of the methods which use elimination to formulate the distributed PGO problem [15, 16, 17], N-ADMM is only outperformed by [17] in terms of accuracy. However, the difference in accuracy is marginal. Lastly, while N-ADMM and ADMM are good choices when accuracy is desired, they are likely to use more time than the methods [15, 16, 17]. Since most methods are not publicly available, the time needed for convergence could not be compared.

It is not clear from the tests on the benchmark datasets what the characteristics of the datasets are for which N-ADMM is most efficient. To gain insight into this matter, this thesis aimed to answer the following sub-question:

"What characterizes the pose graphs for which optimization with N-ADMM is, relative to the current methods, most efficient?"

To answer this question, two models were introduced in Section 5.2. Based on these models, datasets were generated. By varying the parameters of the models, insight was acquired into the effect of these parameters on the efficiency of N-ADMM.

Based on the two models, datasets with a range of graph sizes were created to investigate if there is a correlation between the efficiency of N-ADMM and the size of the graph. To compare its performance, ADMM was run sequentially on the same datasets. First, the block model of Section 5.2.1 was introduced to test for the effect of graph size. In terms of the number of iterations needed for convergence, N-ADMM consistently performed better in comparison to ADMM when the graph size was large. However, this same relation was not found when datasets were created with the Gaussian model of Section 5.2.2. Therefore, no correlation between the relative efficiency of N-ADMM and the size of the graph could be discovered. Possibly, a complexity is created by the block model which propagates per generated pose. This complexity might not be created by the Gaussian model. However, it is unknown what this complexity might be and discovering it might not be simple.

With the Gaussian model, the variance and mean of a relative pose measurement could be varied. By doing so, the effect of bad initial guesses was investigated. With the Gaussian model, a positive correlation was found between the values for the mean and variance and the number of iterations needed for convergence. However, as both N-ADMM and ADMM have similar performance on these datasets, this does not provide insight into the relative efficiency of N-ADMM.

Unfortunately, this report was not able to find the exact scenarios in which N-ADMM is more efficient than the current methods. However, it is clear from the tests in Sections 5.1 and 5.2.1 that such scenarios exist. Further research into how complexity is defined for PGO datasets is necessary. Recommendations for future research into dataset complexity and other directions will be given in the next section.

6.2 Further research

This thesis provided one method to implement Nesterov acceleration in ADMM for distributed PGO. Different choices to implement Nesterov acceleration could be made. Possible paths for research into these choices are

- **Parallelization**
  To bring down computation times, computation can be distributed over the processing units of the robots or over different cores within the same processor. Primal and dual updates will then be performed in parallel by the units. One core or robot would then perform all the primal and dual updates belonging to one subgraph. However, in the current form, the primal updates are performed sequentially in (4.7). When the values $x_g(k+1)$ are calculated, the values $x_i(k+1)$ with $i < g$ are used for this update. When parallel computing is applied, knowledge of these values will be shared after the primal and dual updates and will not be available during these updates. Therefore, algorithm (4.7) must be rewritten to accommodate this. For this purpose, the Jacobi formulation of multi-block ADMM [41] can be used instead of the Gauss-Seidel form that is used in this thesis. Parallelization would provide a significant increase in the efficiency of N-ADMM. However, this may come at the cost of a higher likelihood of divergence. As a stabilizing framework is already used in (4.7), this might not be concerning.

- **Update rules**
  To create N-ADMM, the choice was made in this thesis to use the penalty factor update by [20] instead of the penalty factor update as per [22]. However, this also means that no update rule for σ is present in (4.7). For this reason, $\sigma = 1$ was assumed. The effect of different values for σ could be analysed. Moreover, by adapting the penalty factor update rule of [22] such that $\rho(k)$ will also be decreased, the rule of [22] could be implemented. The value of σ will then be adjusted accordingly.

- **Adaptive M**
  Currently the value M is taken to be constant in (4.7). However, it was clear from the findings in Section 5.1 that different values for M were optimal for each dataset. For some datasets, the condition on the Lagrangians will never be met regardless of the value $\tau(k)$. This $\tau(k)$ was used to balance between the plain iterations and the accelerated iterations. Instead of reducing this value for $\tau(k)$ if the condition is not met, one could also start with the plain iterations and increase the value of $\tau(k)$ if the condition is met. This way, unnecessary steps will be avoided, while acceleration is achieved when possible.

Unfortunately, the exact characteristics of the datasets for which N-ADMM was more efficient, relative to ADMM, were not found in Section 5.2. To attempt to find these characteristics, the recommendation is made to research the following:

- **Loop closures**
  In this thesis, a roughly linear trajectory was simulated. However, the introduction of loop closures is known to complicate scenarios. The N-ADMM algorithm might perform better in these scenarios.

- **Information matrix**
  The choice was made to use the information matrix that is used for every relative pose measurement in the simulated dataset M3500. However, this matrix is not constant in the datasets that have been collected by robots. The effect of the information matrix could be analysed in future work.

- **Outliers**
  No outliers were present in the datasets in Section 5.2. The effect of such outliers can be studied.
- **Experimentation**
  The datasets in Section 5.2 were created through simulation. However, N-ADMM has been shown to perform exceptionally for a dataset which was acquired through experimentation. Collecting the datasets in experimentation could be a promising path for future research.

Furthermore, only two-dimensional datasets were considered in this thesis. The N-ADMM algorithm might perform exceptionally on three-dimensional datasets. Lastly, the choice was made in this thesis to use Nesterov acceleration due to its optimal convergence rate $\mathcal{O}(1/k)$. Other methods for acceleration, e.g. Quasi-Newton schemes, can be investigated.

6.3 Contributions

This thesis has contributed the following:

- **Creation of the N-ADMM algorithm**
  In this thesis, the N-ADMM algorithm (4.7) was created by combining, adapting and extending the current literature and methods to solve the distributed PGO problem. A summary of the adaptations and extensions is listed shortly in Section 6.1 as an answer to the main research question. Details of the N-ADMM algorithm and its creation are given in Chapter 4. It has shown promising results when compared to the plain ADMM iterations [6].
- **Comparison of N-ADMM and ADMM [6] with the state of the art**
  This thesis has provided a comparison of N-ADMM with state of the art methods [6, 15, 16, 17, 14, 13] by running N-ADMM and ADMM [6] sequentially on the benchmark datasets. Furthermore, the performance of ADMM for distributed PGO was compared with the more recent DGS and AMM methods. Lastly, a clear gap between the methods N-ADMM, ADMM and AMM, which solve the nonconvex optimization problem directly, and DGS, which uses approximations to solve distribute PGO, was found.
- **New models for more extensive testing**
  To test for which datasets N-ADMM is more efficient and what these datasets are characterized by, two new models were introduced. The first model created challenging datasets while the second model created a more realistic scenario. While no clear characteristics were found in testing, the first steps towards this goal were made. Furthermore, it provides a more complete picture than the tests on the limited number of publicly available datasets alone.
- **Multi-block application for stabilizing framework [22]**
  In the work [22], an optimization problem of form (3.4) was assumed. The application of the theory of [22] to the multi-block case was given as a recommendation for future work. One such application was given in this thesis and the framework of [22] was adapted and extended accordingly.

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox, *Probalistic robotics.* Emerald Group Publishing Limited, 2006.

[2] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part I,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[3] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): part II,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, “Past, present, and future of simultaneous localization and mapping: toward the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[5] S. Saeedi, M. Trentini, M. Seto, and H. Li, “Multiple-robot simultaneous localization and mapping: a review,” *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[6] S. Choudhary, L. Carlone, H. I. Christensen, and F. Dellaert, “Exactly sparse memory efficient SLAM using the multi-block alternating direction method of multipliers,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1349–1356, Sep. 2015.

[7] S. Thrun, “Simultaneous localization and mapping,” in *Robotics and cognitive approaches to spatial mapping*, pp. 13–41, Springer, 2007.

[8] F. Dellaert and M. Kaess, “Factor graphs for robot perception,” *Foundations and Trends in Robotics*, vol. 6, pp. 1–139, 01 2017.

[9] H. Strasdat, J. Montiel, and A. J. Davison, “Real-time monocular SLAM: why filter?,” in *IEEE International Conference on Robotics and Automation*, pp. 2657–2664, 2010.

[10] G. Stathopoulos, H. Shukla, A. Szucs, Y. Pu, C. N. Jones, *et al.*, *Operator splitting methods in control*, vol. 3. Now Publishers, Inc., 2016.

[11] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[12] L. Carlone and F. Dellaert, "Duality-based verification techniques for 2D SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4589–4596, May 2015.

[13] T. Fan and T. Murphey, "Majorization minimization methods to distributed pose graph optimization with convergence guarantees," *arXiv preprint arXiv:2003.05353*, 2020.

[14] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed trajectory estimation with privacy and communication constraints: a two-stage distributed Gauss-Seidel approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5261–5268, May 2016.

[15] K. Ni and F. Dellaert, "Multi-level submap based SLAM using nested dissection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2558–2565, 2010.

[16] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2D and 3D mapping," in *IEEE International Conference on Robotics and Automation*, pp. 273–278, 2010.

[17] B. Suger, G. D. Tipaldi, L. Spinello, and W. Burgard, "An approach to solving large-scale SLAM problems with a small memory footprint," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3632–3637, 2014.

[18] F. Dellaert, "Factor graphs and GTSAM: a hands-on introduction," tech. rep., Georgia Institute of Technology, 2012.

[19] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g^2o: a general framework for graph optimization," in *IEEE International Conference on Robotics and Automation*, pp. 3607–3613, May 2011.

[20] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[21] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Dokl. Akad. Nauk SSSR*, vol. 269, pp. 543–547, 1983.

[22] A. Themelis, L. Stella, and P. Patrinos, "Douglas–Rachford splitting and ADMM for nonconvex optimization: new convergence results and accelerated versions," *arXiv preprint arXiv:1709.05747*, 2017.

[23] V. Hryhorenko and D. Klyushin, "Multiblock ADMM with Nesterov acceleration," in *XVIII International Conference on Data Science and Intelligent Analysis of Information*, pp. 358–364, Springer, 2018.

[24] G. Calafiore, L. Carlone, and F. Dellaert, "Pose graph optimization in the complex domain: Lagrangian duality, conditions for zero duality gap, and optimal solutions," *arXiv preprint arXiv:1505.03437*, 2015.

[25] A. Alcocer, P. Oliveira, A. Pascoal, and J. Xavier, "Estimation of attitude and position from range-only measurements using geometric descent optimization on the special Euclidean group," in *9th International Conference on Information Fusion*, pp. 1–8, 2006.

[26] R. Tron and R. Vidal, "Distributed 3-D localization of camera sensor networks from 2-D image measurements," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3325–3340, 2014.

[27] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert, "Initialization techniques for 3D SLAM: a survey on rotation estimation and its use in pose graph optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4597–4604, May 2015.

[28] L. Carlone, D. M. Rosen, G. Calafiore, J. J. Leonard, and F. Dellaert, "Lagrangian duality in 3D SLAM: verification techniques and optimal solutions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 125–132, 2015.

[29] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: lightweight algorithms and object-based models," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017.

[30] R. Hartley, J. Trumpf, Y. Dai, and H. Li, "Rotation averaging," *International Journal of Computer Vision*, vol. 103, no. 3, pp. 267–305, 2013.

[31] D. Martinec and T. Pajdla, "Robust rotation and translation estimation in multiview reconstruction," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

[32] J. Fredriksson and C. Olsson, "Simultaneous multiple rotation averaging using Lagrangian duality," in *Asian Conference on Computer Vision*, pp. 245–258, Springer, 2012.

[33] G. C. Sharp, S. W. Lee, and D. K. Wehe, "Multiview registration of 3D scenes by minimizing error between coordinate frames," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 1037–1050, 2004.

[34] G. Dubbelman, P. Hansen, B. Browning, and M. B. Dias, "Orientation only loop-closing with closed-form trajectory bending," in *IEEE International Conference on Robotics and Automation*, pp. 815–821, 2012.

[35] D. F. Glas, T. Miyashita, H. Ishiguro, and N. Hagita, "Automatic position calibration and sensor displacement detection for networks of laser range finders for human tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2938–2945, 2010.

[36] V. M. Govindu, "Combining two-view constraints for motion estimation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 218–225, 2001.

[37] F. Dellaert, J. Carlson, V. Ila, K. Ni, and C. E. Thorpe, "Subgraph-preconditioned conjugate gradients for large scale SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2566–2571, 2010.

[38] Y.-D. Jian and F. Dellaert, "iSPCG: incremental subgraph-preconditioned conjugate gradient method for online SLAM with many loop-closures," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2647–2653, 2014.

[39] N. Parikh, S. Boyd, *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[40] S. Boyd and L. Vandenberghe, *Convex Optimization.* USA: Cambridge University Press, 2004.

[41] W. Deng, M.-J. Lai, Z. Peng, and W. Yin, "Parallel multi-block ADMM with O(1/k) convergence," *Journal of Scientific Computing*, vol. 71, no. 2, pp. 712–736, 2017.

[42] B. He, M. Tao, and X. Yuan, "Alternating direction method with Gaussian back substitution for separable convex programming," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 313–340, 2012.

[43] B. He, L. Hou, and X. Yuan, "On full Jacobian decomposition of the augmented Lagrangian method for separable convex programming," *SIAM Journal on Optimization*, vol. 25, no. 4, pp. 2274–2312, 2015.

[44] L. Carlone, "A convergence analysis for pose graph optimization via Gauss-Newton methods," in *IEEE international conference on robotics and automation*, pp. 965–972, 2013.

[45] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[46] Y. Wang, W. Yin, and J. Zeng, "Global convergence of ADMM in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, vol. 78, no. 1, pp. 29–63, 2019.

[47] A. Themelis and P. Patrinos, "Douglas–Rachford splitting and ADMM for nonconvex optimization: tight convergence results," *SIAM Journal on Optimization*, vol. 30, no. 1, pp. 149–181, 2020.

[48] C. Chen, B. He, Y. Ye, and X. Yuan, "The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent," *Mathematical Programming*, vol. 155, no. 1-2, pp. 57–79, 2016.

[49] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pp. 28–28, IEEE, 1998.

[50] L. Carlone and A. Censi, "From angular manifolds to the integer lattice: guaranteed orientation estimation with application to pose graph optimization," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 475–492, 2014.

Appendix A

Graphical PGO example

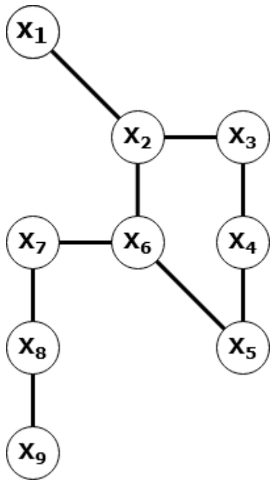


Figure A.1: Simple pose graph with poses x_i as nodes and measurements as edges.

A simple pose graph optimization (PGO) example is given in this appendix. Here, only a graphical representation will be given. For the mathematical formulation of PGO, the reader is referred to Chapter 2.

Consider a simple simultaneous localization and mapping (SLAM) scenario where a robot moves along a trajectory in an unknown environment. In this scenario, a robot is equipped with odometry sensors and a laser range finder. Based on the measurements from these sensors, the pose graph will be constructed. The true position of the robot will never be known in SLAM problems. However, a good estimate can be made through PGO. A pose graph is constructed through nodes and edges. Nodes correspond to the poses that make up the trajectory of the robot and edges correspond to the measurements. An example of a pose graph with trajectory $[x_1, x_2, \ldots, x_9]$ is depicted in Figure A.1. A measurement x_{ij} connects pose x_i to pose x_j in this trajectory. A special measurement known as a loop closure is present between poses x_2 and x_6. These connections are made when the robot recognizes a location it has previously visited. Since the true poses are never known in the SLAM problem, an initial guess must be made to construct the initial pose graph. These initial guesses will move closer to the true poses through optimization.

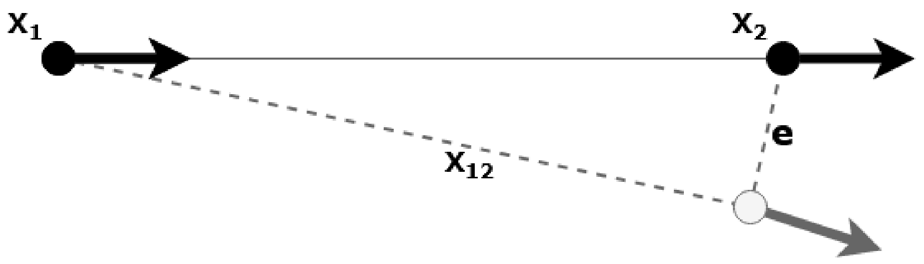


Figure A.2: Two consecutive poses x_1 and x_2 of the pose graph with measurement x_{12}. The fictional measured pose is shown in grey. The error between the initial guess and the measurement is denoted with e.

In Figure A.2, two consecutive poses of the pose graph in Figure A.1 are depicted. To create the pose graph an initial guess of pose x_2 is made relative to pose x_1. This initial guess is usually based on the odometry data alone. The measurement from the laser range finder is decoded in the relative pose measurement x_{12}. By minimizing the error e, an estimation will be made of the true pose. In the case of Figure A.2, the pose will correspond with the measurement after estimation. However, pose graphs consist of a chain of elements as in Figure A.2. To estimate the full trajectory, all errors in the pose graph will have to be minimized. If the error is reduced in one element of the pose graph, the error in another element might be increased. Therefore, the estimated trajectory will be the result of a consensus where the sum of all errors is as small as possible.

In PGO, multiple sensors can encode the relative pose measurement. Furthermore, landmarks can easily be included in this framework. Landmarks will take a similar role as a pose in this case. The SLAM problem can then be solved by not only estimating a trajectory, but also estimating the location of the landmarks. For an extensive overview of PGO in SLAM problems, the reader is referred to [8].

Appendix B

Supporting results and figures

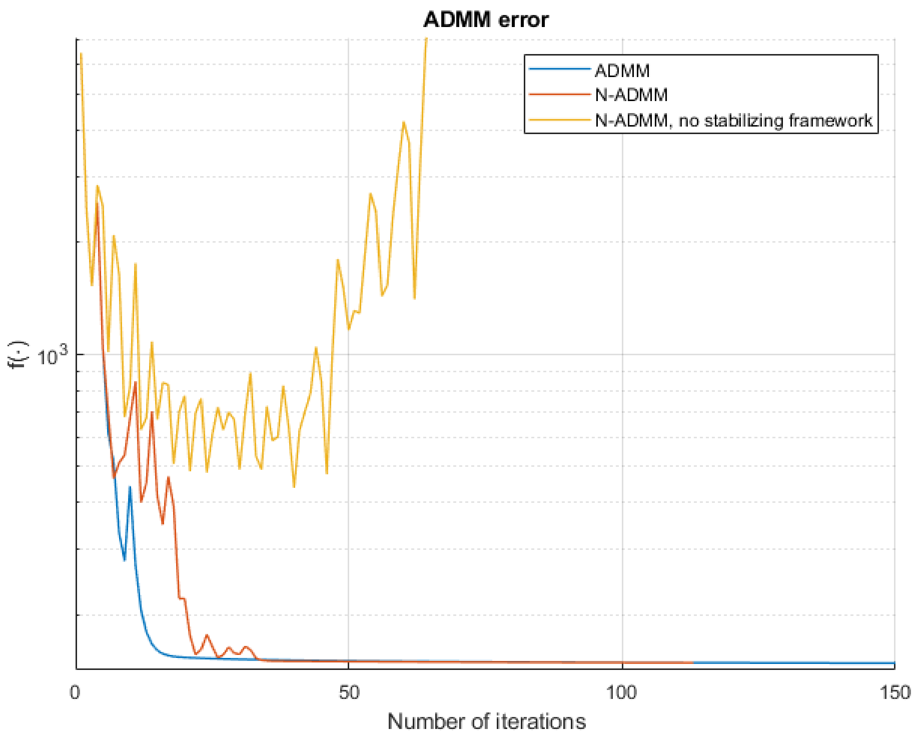


Figure B.1: Convergence of ADMM and N-ADMM. Divergence for N-ADMM without stabilizing framework. Here, $M = 3, N = 10, \rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$. Tests were performed on the dataset M3500

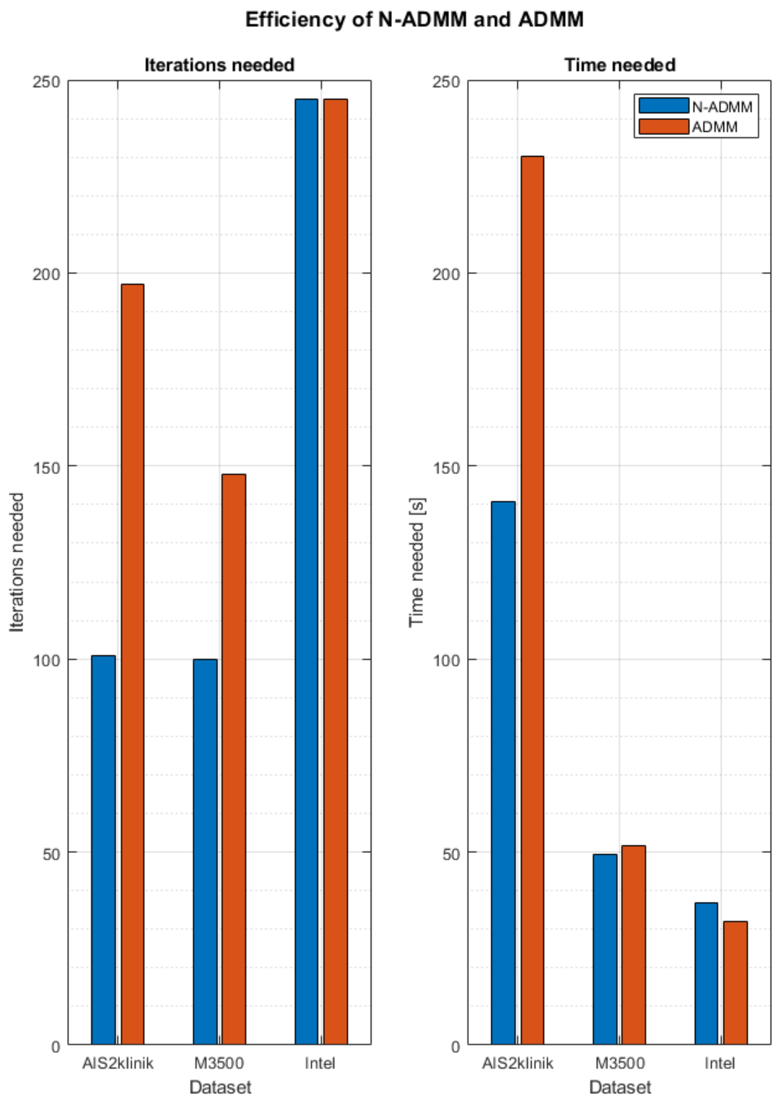


Figure B.2: Iterations and time needed for N-ADMM and ADMM. Acquired with $N = 10,\ \rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$. Graphical representation of Table 5.3.

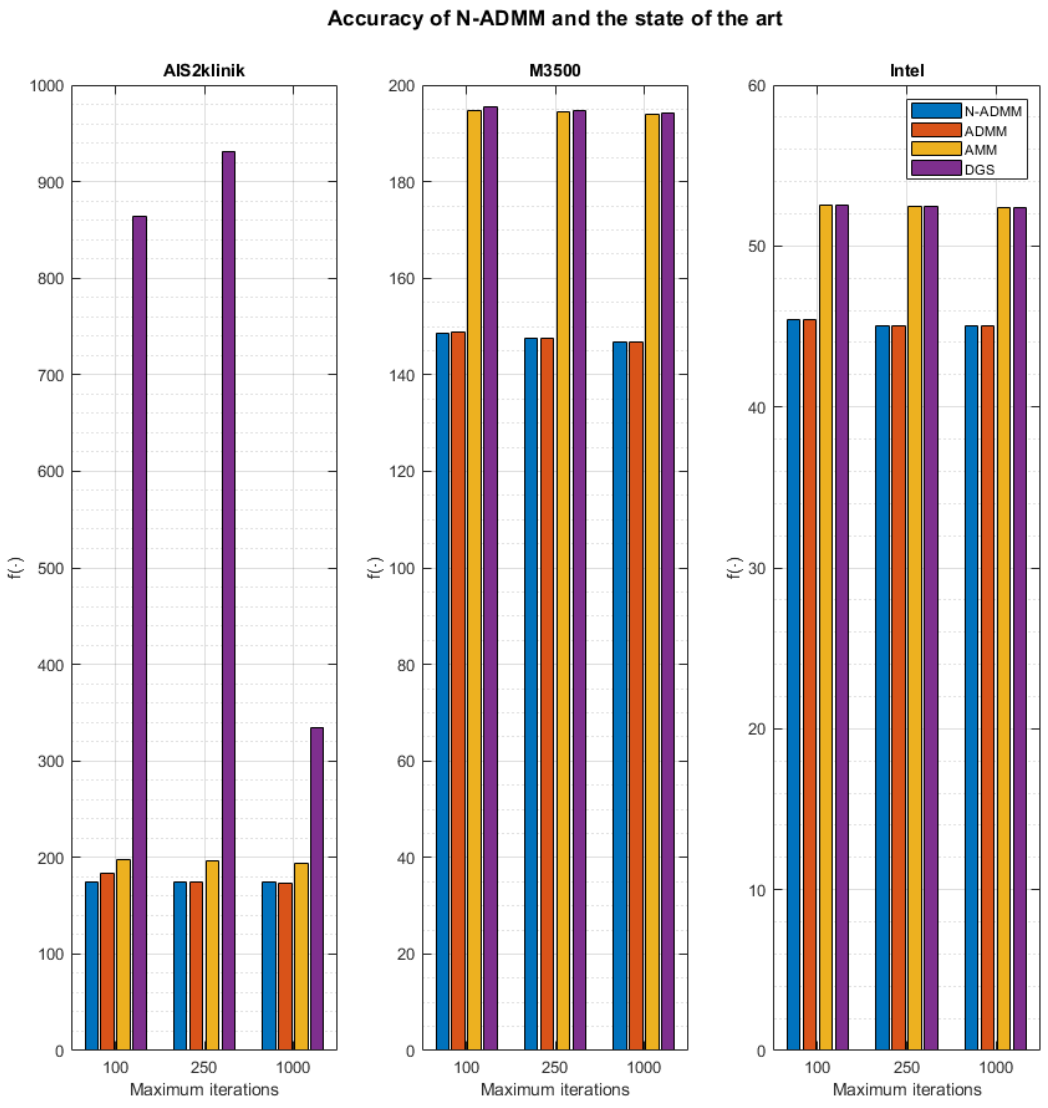


Figure B.3: Values of $f(\cdot)$ at $k = \{100, 250, 1000\}$. Results achieved with $M = 3, N = 10$ and $\rho(0) = 0.2$. Graphical representation of Table 5.4.

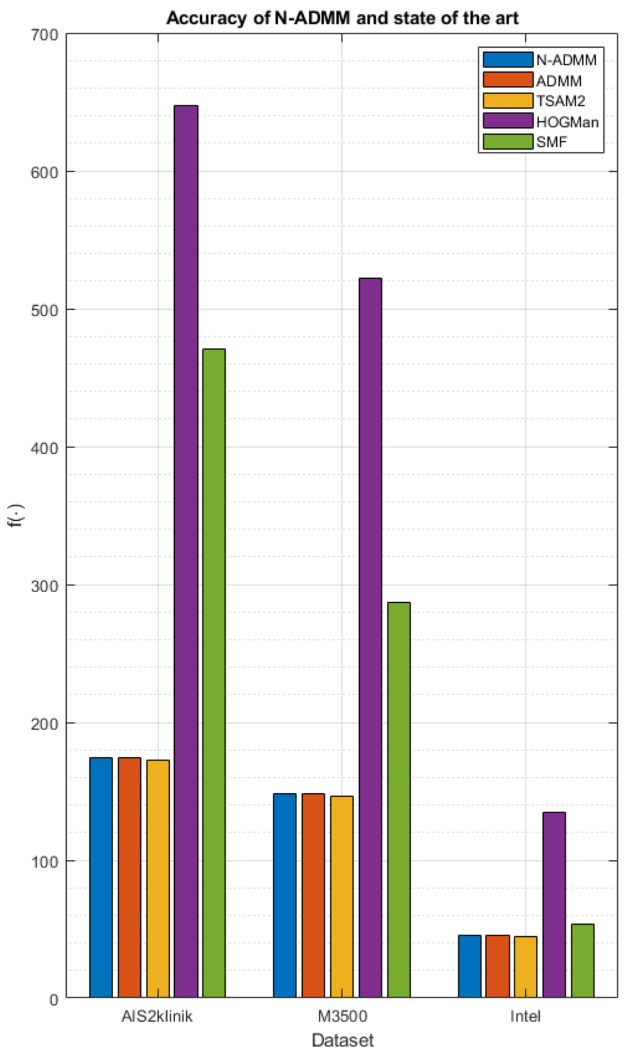


Figure B.4: Values of $f(\cdot)$ at $k = 200$. Results acquired with $M = 3,\ N = 10,\ \rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$. Graphical representation of Table 5.5.

Table B.1: Complete table for varying M. Here, $N = 10,\ \rho(0) = 0.2$ and stopping conditions $p_{res} < 0.1$ and $d_{res} < 0.1$.

Dataset	M	Iterations needed		time [s]	
		N-ADMM	ADMM	N-ADMM	ADMM
intel	1	245	245	36.97	32.03
	2	245	245	42.28	32.19
	3	245	245	47.69	31.92
	4	245	245	51.68	32.01
	5	245	245	56.55	31.82
m3500_g2o	1	130	148	55.25	53.29
	2	130	148	56.50	51.39
	3	112	148	52.36	51.34
	4	100	148	49.49	51.71
	5	113	148	60.94	51.20
ais2klinik	1	101	197	140.91	230.25
	2	101	197	149.29	231.62
	3	115	197	179.40	229.31
	4	113	197	183.42	229.48
	5	113	197	193.18	232.02