UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

Master's degree in Control Systems Engineering

20 October 2023

Distributed hybrid unit quaternion localisation of camera networks

Supervisor:
Prof. CENEDESE ANGELO
Co-Supervisor:
Prof. ZACCARIAN LUCA

Candidate:
CALLEGARI SARA
2020373

ACADEMIC YEAR 2022/2023

To my mum, my biggest supporter, who always believed in me
To my family, always there for me

To my oldest friends, who decided to be part of my life
To the UniPd ones, that made these years unforgettable

To Toulouse, a place that holds some of the best memories of my life
To the new experiences that this place gifted me
To the LAAS gang and the amazing Erasmus people

To Yoni, for his patience, understanding, and endless motivation

A thank you to my supervisor Prof. Angelo Cenesede
For his teaching and mentorship

A thank you to my co-supervisor Prof. Luca Zaccarian
For his continuous support and the opportunity he provided me

Abstract

This thesis presents a framework for distributed localization of camera sensor networks, based on the use of quaternions.

We begin by providing a brief overview of the fundamental tools needed for understanding our framework, introducing key elements of the theory, such as *quaternions' properties* and *graph theory*, as well as some details about the *estimation problem* in the *camera calibration* framework, and an analysis of *hybrid dynamical systems*.

Then we include an overview of two approaches already present in the literature, that inspired this work.

Lastly, we introduce our proposed model and we show its effectiveness and robustness with respect to various parameters.

Contents

Chapter 1

Introduction

Several dynamical systems evolve on angular type of variables, such as the pose of rigid bodies. Perhaps the most suitable mathematical tool for describing such dynamics corresponds to the n-dimensional sphere, that is the manifold of dimension n embedded in the (n+1) dimensional Euclidean space and corresponding to all the vectors having unit norm. A relevant example corresponds to the 3-sphere and the ensuing quaternion-based coordinate system, which is largely used for describing the pose of rigid bodies. One of the challenges in describing dynamics evolving on the n-dimensional sphere is the fact that global robust stabilization of a point cannot be accomplished with continuous feedback laws. It is then necessary to resort to alternative solutions, to ensure robustness of the closed-loop stability properties. Hybrid dynamical systems [1] are a possible answer to this and will be investigated in this thesis, where existing works on the distributed calibration of camera networks will be first overviewed, and hybrid solutions will be proposed and tested.

In camera networks, the distributed calibration problem refers to the challenge of achieving accurate and consistent calibration of multiple cameras in the network. Camera calibration is the process of determining the internal and external parameters of a camera that are necessary for accurate measurements and 3D reconstruction from its 2D image data. In a camera network, there can be multiple cameras deployed at different locations and orientations. To ensure accurate reconstruction and measurement across multiple cameras, it is essential to calibrate them. The distributed calibration problem arises when the calibration process needs to be performed without a centralized authority or a single point of reference. This can occur in scenarios where cameras are deployed in a distributed manner, such as in surveillance systems [2], smart environments [3], or robotic applications [4]. Several factors contribute to the complexity of the distributed calibration problem such as lack of global knowledge, heterogeneous camera characteristics, and communication and synchronization constraints. Solving the distributed calibration problem requires developing al-

gorithms and techniques that can handle these challenges.

Distributed calibration algorithms aim to iteratively refine camera parameters based on local information and limited communication, gradually aligning the calibration of cameras across the network. Techniques like bundle adjustment, self-calibration, and consensus-based estimation are commonly used to tackle this problem. Overall, the distributed calibration problem in camera networks is a complex task that requires robust algorithms to achieve accurate and consistent calibration across multiple cameras in a distributed and decentralized environment.

We address the distributed calibration of Camera Sensor Networks using the quaternion representation to describe the pose of the cameras in the network, we exploit this representation by taking inspiration from [5]. A similar approach is used in [6], where double quaternions are exploited to address the calibration problem.

The starting point of the work is the Tron-Vidal approach [7], also called *TV algorithm*. The biggest innovation proposed in this paper is that the optimization in the manifold $\mathbb{SE}(3)$ is split into two steps, since $\mathbb{SE}(3) = \mathbb{SO}(3) \times \mathbb{R}^3$. The first step consists of the estimation of the rotational part of the absolute pose, computing a Riemannian gradient descent on $\mathbb{SO}(3)$; secondly, the translations are estimated computing a gradient descent step on the Euclidian space $\mathbb{R}^3$. The main drawbacks of this approach are that it is able to find a good solution only if the first estimate of the camera's attitude is sufficiently close to the real one and that the estimation error is not distributed equally among the two pose components. In fact, being the translation estimated after the rotation the error accumulated is much greater.

The envisaged steps are as follows:

- Study of state of art and of Matlab implementation of TV algorithm;
- Definition of a new estimation law, using unit quaternions, and subsequent implementation;
- Definition of a hybrid law for the further minimization of the cost function.

Chapter 2

Background material

We introduce briefly some background material that is needed to understand better the next chapters. Here, you will find a concise overview of key concepts related to quaternions, graph theory, camera calibration, and hybrid dynamical systems.
These notions will help to better understand the approaches taken from the literature on distributed calibration and the new approach that we will introduce later.

2.1 Quaternion Basics: Definitions and Properties

We summarize here some definitions and properties of quaternion [8], that will be useful later.
A quaternion represents an extension of a complex number in a higher dimensional space, hence it is a hyper-complex entity represented by four parameters, the real one η and the complex part $\boldsymbol{\epsilon}$:

$$q = \eta + x\epsilon_x + y\epsilon_y + z\epsilon_z = \eta + \boldsymbol{\epsilon} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} \tag{2.1}$$

A quaternion is called of **unity norm** if $\|q\|^2 = 1$, hence:

$$\|q\|^2 = \eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = \eta^2 + \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = 1 \tag{2.2}$$

For a unit quaternion we can write:

$$q = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \mathbf{u}\sin(\frac{\theta}{2}) \end{bmatrix} \in \mathbb{S}^3 \tag{2.3}$$

We can define the **Rodriguez** rotation formula for unit quaternions, thanks to which we can relate matrix and quaternion rotations.

$$R(q) = \mathrm{I}_3 + 2\eta \boldsymbol{S}(\boldsymbol{\epsilon}) + 2\boldsymbol{S}^2(\boldsymbol{\epsilon}) \quad \text{with} \quad \boldsymbol{S}(\boldsymbol{\epsilon}) = \begin{bmatrix} 0 & -\epsilon_z & \epsilon_y \\ \epsilon_z & 0 & -\epsilon_x \\ -\epsilon_y & \epsilon_x & 0 \end{bmatrix} \tag{2.4}$$

$$= \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z + \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y + \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z - \eta\epsilon_y) & 2(\epsilon_y\epsilon_z + \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix} \tag{2.5}$$

From (2.4) we can observe the **double coverage** property of quaternions, in fact, $R(q) = R(-q)$, so the rotation matrix R is associated with two different quaternions: $q = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix}$ and $-q = \begin{bmatrix} -\eta \\ -\boldsymbol{\epsilon} \end{bmatrix}$.

The **Hamilton product**, or quaternion multiplication, is the product of two quaternions that gives as a result another quaternion, hence $\mathbb{S}^3 \times \mathbb{S}^3 \to \mathbb{S}^3$. It is defined as follows:

$$q_1 \odot q_2 = \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\epsilon}_1^T\boldsymbol{\epsilon}_2 \\ \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 \end{bmatrix} \tag{2.6}$$

$$= \begin{bmatrix} \eta_1 & -\boldsymbol{\epsilon}_1^T \\ \boldsymbol{\epsilon}_1 & \eta_1\mathrm{I}_3 + \boldsymbol{S}(\boldsymbol{\epsilon}_1) \end{bmatrix} \begin{bmatrix} \eta_2 \\ \boldsymbol{\epsilon}_2 \end{bmatrix} = F(q_1)q_2 \quad \text{with } F(q) = \begin{bmatrix} \eta & -\boldsymbol{\epsilon}^T \\ \boldsymbol{\epsilon} & \eta\mathrm{I}_3 + \boldsymbol{S}(\boldsymbol{\epsilon}) \end{bmatrix} \tag{2.7}$$

$$= \begin{bmatrix} \eta_2 & -\boldsymbol{\epsilon}_2^T \\ \boldsymbol{\epsilon}_2 & \eta_2\mathrm{I}_3 - \boldsymbol{S}(\boldsymbol{\epsilon}_2) \end{bmatrix} \begin{bmatrix} \eta_1 \\ \boldsymbol{\epsilon}_1 \end{bmatrix} = G(q_2)q_1 \quad \text{with } G(q) = \begin{bmatrix} \eta & -\boldsymbol{\epsilon}^T \\ \boldsymbol{\epsilon} & \eta\mathrm{I}_3 - \boldsymbol{S}(\boldsymbol{\epsilon}) \end{bmatrix} \tag{2.8}$$

The quaternion **inverse** is defined as:

$$q^{-1} = \begin{bmatrix} \eta \\ -\boldsymbol{\epsilon} \end{bmatrix} \tag{2.9}$$

And we have a **null rotation** if we combine a rotation with its inverse:

$$q \odot q^{-1} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} \odot \begin{bmatrix} \eta \\ -\boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} = q_I \tag{2.10}$$

Notice also that:

$$F^T(q) = \begin{bmatrix} \eta & \boldsymbol{\epsilon}^T \\ -\boldsymbol{\epsilon} & \eta \mathrm{I}_3 - \boldsymbol{S}(\boldsymbol{\epsilon}) \end{bmatrix} = F(q^{-1}) \tag{2.11}$$

$$G^T(q) = \begin{bmatrix} \eta & \boldsymbol{\epsilon}^T \\ -\boldsymbol{\epsilon} & \eta \mathrm{I}_3 + \boldsymbol{S}(\boldsymbol{\epsilon}) \end{bmatrix} = G(q^{-1}) \tag{2.12}$$

In rigid body motion, the variation of q along a motion with angular velocity $\omega \in \mathbb{R}^3$ linearly depends on ω. Thus, the quaternion **derivative** results in:

$$\dot{q}(t) = \frac{1}{2} q \odot \begin{bmatrix} 0 \\ \omega \end{bmatrix} \tag{2.13}$$

$$= \frac{1}{2} F(q) \begin{bmatrix} 0 \\ \omega \end{bmatrix} \tag{2.14}$$

$$= \frac{1}{2} F_2(q)\omega \quad \text{with } F_2(q) = \begin{bmatrix} -\boldsymbol{\epsilon}^T \\ \eta \mathrm{I}_3 + \boldsymbol{S}(\boldsymbol{\epsilon}) \end{bmatrix} \tag{2.15}$$

The **distance** between two quaternions may be defined as follows:

$$d(q_1, q_2) = (q_1^{-1} \odot q_2)^T \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathrm{I}_3 \end{bmatrix} (q_1^{-1} \odot q_2) \tag{2.16}$$

$$= (q_1^{-1} \odot q_2)^T \begin{bmatrix} \mathbf{0} \\ \mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} (q_1^{-1} \odot q_2) \tag{2.17}$$

$$= \left| \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(q_1^{-1}) q_2 \right|^2 \tag{2.18}$$

$$= \left| \begin{bmatrix} -\boldsymbol{\epsilon}_1 & \eta_1 \mathrm{I}_3 - \boldsymbol{S}(\boldsymbol{\epsilon}_1) \end{bmatrix} \begin{bmatrix} \eta_2 \\ \boldsymbol{\epsilon}_2 \end{bmatrix} \right|^2 \tag{2.19}$$

$$= \left| \eta_1 \boldsymbol{\epsilon}_2 - \eta_2 \boldsymbol{\epsilon}_1 - \boldsymbol{S}(\boldsymbol{\epsilon}_1)\boldsymbol{\epsilon}_2 \right|^2 \tag{2.20}$$

$$= \left| -\eta_1 \boldsymbol{\epsilon}_2 + \eta_2 \boldsymbol{\epsilon}_1 + \boldsymbol{S}(\boldsymbol{\epsilon}_1)\boldsymbol{\epsilon}_2 \right|^2 \tag{2.21}$$

$$= \left| \eta_2 \boldsymbol{\epsilon}_1 - \eta_1 \boldsymbol{\epsilon}_2 - \boldsymbol{S}(\boldsymbol{\epsilon}_2)\boldsymbol{\epsilon}_1 \right|^2 \tag{2.22}$$

$$= \left| \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(q_2^{-1}) q_1 \right|^2 \tag{2.23}$$

From the above derivations, we immediately obtain:

$$d(q_1, q_2) = d(q_1^{-1}, q_2^{-1}) = d(q_2, q_1) = d(q_2^{-1}, q_1^{-1}) \tag{2.24}$$

2.2 Elements of Graph Theory

We represent a network of N nodes as a directed, symmetric, and connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [9].
A graph is defined by a set $\mathcal{V} = v_1, ..., v_N$ of vertices that represent the nodes in the network and by a set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of edges that indicate pairs of nodes that can interact with each other, i.e., that can communicate or that can measure their relative rotation and translation. These sets are disjoint.

We consider finite graphs only, meaning that the sets $\mathcal{V}$ and $\mathcal{E}$ are both always finite.
We use single indexes for nodes (i.e., 'i' for $i \in \mathcal{V}$) and double indexes for edges (i.e., 'ij' for $(i, j) \in \mathcal{E}$).

The set of vertices adjacent to a vertex $i \in \mathcal{G}$ is said to be the neighbourhood of i. Given the graph $\mathcal{G}$, we define its adjacency matrix $A = A_{\mathcal{G}} = (a_{ij})$. The matrix $A_{\mathcal{G}} \in \mathbb{R}^{n \times n}$ is defined as follows, for an undirected graph:

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

2.3 Camera Calibration

When we talk about camera calibration we mean the essential process in visual servoing that involves determining the intrinsic and extrinsic parameters of a camera system, which are essential for accurately relating the 2D image coordinates of objects in the camera's field of view to their corresponding 3D world coordinates. It is a crucial aspect of computer vision and robotics, particularly in applications where a camera is used to guide or control the motion of a robotic system.

The internal parameters of cameras are their own attributes. In traditional visual calibration theory, it is generally considered that intrinsic parameters are unchanged, which includes effective focal length, optical center, image coordinate origin, non-vertical factor, radial distortion, and tangential distortion. The external parameters of cameras are the mapping relationship between the 3-dimensional world and the image coordinate system, which consists of a rotation matrix and a translation vector.
More in detail we have:

- Intrinsic parameters: Intrinsic parameters characterize the camera's internal properties and distortions. These parameters include:
  - Focal Length (f): The focal length determines how much the camera

lens converges or diverges incoming light rays, affecting the field of view and magnification;

- Principal Point (c_x, c_y): The principal point represents the optical center of the image sensor, where the optical axis intersects it. It is essential for mapping image coordinates to the camera's optical axis;
- Lens Distortion Parameters(k, s, p): Cameras often introduce distortion due to the shape of the lens. Two very common distortion models are radial distortion and tangential distortion. It is fundamental to account for these if we want to have an accurate and undistorted image;

- Extrinsic Parameters: Extrinsic parameters describe the camera's position and orientation in the world coordinate system. They define the transformation matrix between the camera's coordinate frame and the world coordinate frame. These parameters include:
  - Rotation Matrix (R): Describes the orientation of the camera with respect to the world coordinate system;
  - Translation Vector (t): Specifies the position of the camera's optical center in the world coordinate system;

Camera calibration techniques can be divided into three main categories: traditional calibration method, self-calibration method, and calibration method based on active vision.[10] [11]

The *traditional method of camera calibration* is known to calculate intrinsic and extrinsic parameters of cameras through mathematical transformations, given correspondences between points in the image (or images) and actual coordinates of these points in 3D reality. The parameters are then obtained using an optimization algorithm.
The traditional methods can use any camera model and have a high precision of calibration, so when applications require high accuracy, this approach is often used. The typical representatives are as follows: the direct linear transformation method, nonlinear optimization method, two-step method, planar template method, biplane method, and so on.
This category of methods is considered the most accurate but also requires the most manual work to obtain the point correspondences data.

The need for a *self-calibration method* arises when the camera can not be calibrated by choosing the appropriate calibration object.
This method does not depend on the calibration reference object of the camera and is unrelated to the scenes and camera movements, only making use of the self-constraints of camera intrinsic parameters. It is a very flexible method, but the mathematical computation process of the self-calibration method is complex

and is only applicable to situations where less precision is accepted, such as virtual reality applications.
Self-calibration techniques can be divided into: Based on Active Vision camera self-calibration techniques, using essential matrix $E = [t]_x R$ and fundamental matrix $F = k_b^{-T} E k_a^{-1}$ self-calibration method (k_a and k_b are the intrinsic parameter matrices of camera a and b respectively), using the absolute conic and polar transform the nature of camera self-calibration method, using blanking points or blanking line camera calibration method and calibration method under considering the case of non-linear distortion of camera.

The *active calibration method* is a special class of self-calibrated methods. The methods require multiple images acquired by a moving camera with controlled and known motion and perform analysis based on the known motion track of the camera and corresponding images. Even though it is relatively accurate, it has high requirements of cost and equipment to track the camera motion accurately.
The active calibration method has the advantage of a simple algorithm and the disadvantage of that the camera movement can not be applied beyond the control of an unknown or occasion.
Calibration based on three-orthogonal translational motion and the orthogonal motion method based on planar homography matrix are both examples of this category.

The distributed camera calibration problem in visual servoing is an extension of the traditional camera calibration problem that arises when multiple cameras or sensors are used in a distributed or networked manner to guide or control a robotic system. In this scenario, the goal is to determine the intrinsic and extrinsic parameters of each camera or sensor in the network, allowing for a coordinated and accurate perception of the robot's environment. Distributed camera calibration is essential for multi-camera visual servoing systems to ensure that the information obtained from each camera is adequately synchronized and aligned, enabling effective robot control and decision-making.

2.3.1 Estimation problem

Regarding the camera calibration problem, we will focus on the estimation part. Namely, we want to estimate the rotation and translation of each camera. To simplify the problem we will focus on the rotation estimation only.
Instead of using the rotation matrices to represent the problem, we will use quaternions
Let us start with a simple case in which the network is made of only two nodes. We need to consider absolute (with respect to a world frame) and relative rotations.

We can define the absolute rotations as:

$$q_0 = [1 \quad 0 \quad 0 \quad 0]^T \xrightarrow{\text{rot. equiv.}} R_0 = \boldsymbol{I}_3$$
$$q_1 \to R_1$$

Recalling that $q_{ij} = q_j \odot q_i^{-1}$, and similarly $R_{ij} = R_i^T R_j$; we can define also the relative ones:

$$q_{01} \to R_{01}$$
$$q_{10} \to R_{10}$$

Hence the objective rotation is calculated as

$$R_1^* = \text{argmin}_{R_1}(J)$$

where J = suitable average of $(R_0 R_1^T R_{10}, R_1 R_0^T R_{01})$ is the cost function .
If

$$size(q) = d\left(\begin{bmatrix}\eta \\ \boldsymbol{\epsilon}\end{bmatrix}, \begin{bmatrix}1 \\ \mathbf{0}\end{bmatrix}\right) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = \sin^2\frac{\theta}{2} = |q_1|_{\boldsymbol{\epsilon}}^2 = q^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} q \tag{2.26}$$

then the cost function J can be defined as follows

$$J = d(q_1^{-1} \odot q_{10} + d(q_1 \odot q_{01}) \tag{2.27}$$
$$= |G(q_{10})q_1^{-1}|_{\boldsymbol{\epsilon}}^2 + |G(q_{01})q_1|_{\boldsymbol{\epsilon}}^2 \tag{2.28}$$
$$= |F(q_{10}^-1)q_1|_{\boldsymbol{\epsilon}}^2 + |G(q_{01})q_1|_{\boldsymbol{\epsilon}}^2 \tag{2.29}$$

The derivative of the cost function is

$$\dot{J} = 2(F(q_{10}^{-1})q_1)^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} F(q_{10}^{-1})\dot{q}_1 + 2(G(q_{01})q_1)^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} G(q_{01})\dot{q}_1 \tag{2.30}$$
$$= 2\left(q_1^T F(q_{10}^{-1})^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} F(q_{10}^{-1}) + q_1^T G(q_{01})^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} G(q_{01})\right)\dot{q}_1 \tag{2.31}$$
$$= 2\left(F(q_{10}^{-1})^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} F(q_{10}^{-1})q_1 + G(q_{01})^T\begin{bmatrix}0 & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{I}_3\end{bmatrix} G(q_{01})q_1\right)^T\dot{q}_1 \tag{2.32}$$
$$= 2M^T(q_1, q_{01}, q_{10})\dot{q}_1 \tag{2.33}$$
$$\overset{2.15}{=} 2M^T(q_1, q_{01}, q_{10})\frac{1}{2}F_2(q_1)\omega_1 \tag{2.34}$$
$$= M^T(q_1, q_{01}, q_{10})F_2(q_1)\omega_1 \tag{2.35}$$

Where $\omega_1 = k(q_1, q_{01}, q_{10})$ is given by the feedback.
In order to have stability and robustness we need negative semi-definiteness: $\omega_1 = -\mu F_2(q_1)^T M(q_1, q_{01}, q_{10})$, with μ arbitrary gain.

2.4 Introduction to Hybrid Dynamical Systems

Dynamical systems are commonly classified into two fundamental categories: continuous-time and discrete-time dynamical systems. Extensive research has been dedicated to the investigation of these two classes within the academic literature, often in parallel streams of inquiry [1]. Nonetheless, there exists a subset of dynamical systems that manifest characteristics of both continuous and discrete behavior. Examples of such hybrid systems include digitally controlled mechanical apparatuses or electronic circuits that seamlessly integrate analog and digital components. Moreover, when it comes to mathematical modeling, reliance solely on differential and difference equations proves inadequate, particularly in scenarios involving phenomena such as impacts and event-triggered dynamics. Hence hybrid dynamical systems are a class of mathematical models used to describe and analyze complex systems that exhibit both continuous and discrete behaviors.
These systems combine elements of continuous-time dynamics, characterized by differential equations $\dot{x} = f(x),\ x \in \mathbb{R}^n$, and discrete-event dynamics, characterized by first-order equations $x^+ = g(x),\ x \in \mathbb{R}^n$.

A formal framework for characterizing hybrid dynamical systems is distinguished by its amalgamation of continuous-time and discrete-time dynamics, which can be represented as:

$$\mathcal{H} \begin{cases} \dot{x} = f(x), & x \in \mathcal{C} \\ x^+ = g(x), & x \in \mathcal{D} \end{cases} \tag{2.36}$$

The expression 2.36 delineates that the state of the hybrid system denoted as x, exhibits two modes of evolution: one governed by a differential equation, $\dot{x} = f(x)$, while residing in the set $\mathcal{C}$, and another dictated by a difference equation, $x^+ = g(x)$, when situated within the set $\mathcal{D}$.
Here, the symbol $\dot{x}$ signifies the continuous-time derivative of x, while x^+ designates the state's value following a discrete transition, or jump.

Key components of hybrid dynamical systems $\mathcal{H}$ include:

- $\mathcal{C}$ constitutes the flow set: It designates the region where solutions are allowed to evolve continuously (flow);
- f represents the flow map: It characterizes how the state evolves along a continuous trajectory;
- $\mathcal{D}$ constitutes the jump set: This is the realm in which solutions can evolve discretely (jump);
- g represents the jump map: It specifies how the state changes across discrete transitions.

It is noteworthy that the formalism presented in equation 2.36 can be effectively employed to encapsulate classical continuous-time dynamical systems (when $\mathcal{C} = \mathbb{R}^n$ and $\mathcal{D} = 0$) as well as discrete-time dynamical systems (when $\mathcal{C} = 0$ and $\mathcal{D} = \mathbb{R}^n$), thereby encompassing these well-established paradigms as special cases within this comprehensive framework.

A simple example to understand better how hybrid systems work is the thermostat one (Fig.2.1).

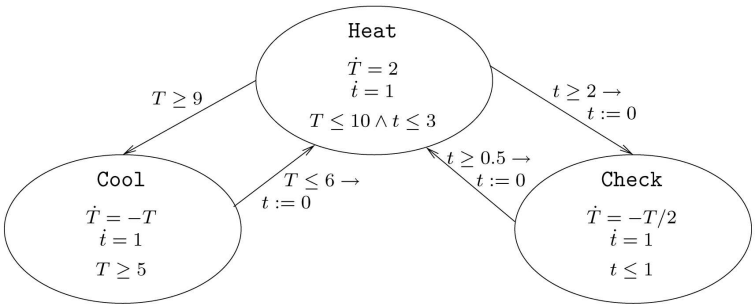


Figure 2.1: Simple model of a thermostat as a hybrid system (image taken from [12])

The thermostat has three locations. The first two, **Heat** and **Cool**, represent states in which the thermostat heats and cools the environment it operates in, respectively. The third, **check**, is a self-diagnostic state. The continuous state space of the thermostat contains two continuous variables, namely a clock $t \in \mathbb{R} \geq 0$ and a temperature $T \in \mathbb{R} \geq 0$.
We can limit the continuous state-space, such that both the clock t and the temperature T are within an interval, hence the continuous state is $(t, T) \in [0, 100]^2$.
Each location has an associated invariant predicate defining the set of permitted values for the continuous variables while in that location. The invariants for the thermostat are:

$$\begin{aligned}
&\text{Inv}_{Heat}(t, T) := T \leq 10 \wedge t \leq 3 \\
&\text{Inv}_{Cool}(t, T) := T \geq 5 \\
&\text{Inv}_{Check}(t, T) := t \leq 1
\end{aligned}$$

The initial states of a hybrid system are determined by a predicate Initial condition. For the thermostat, $Init(l, t, T)$ is defined as $l = Heat \wedge t = 0 \wedge 5 \leq T \leq 10$.
The discrete transitions between locations describe the logic of the software system. Each such transition is comprised of two components: a flow predicate defining a subset of the continuous state space in which the transition is enabled

(permitted), and a jump function describing an instantaneous change applied as a side effect of the transition.

The invariant in the Heat location is $T \leq 10 \wedge t \leq 3$, that is, the system cannot remain in this location when the temperature exceeds ten and the clock exceeds three-time units. The control can switch to the Cool location, which models that the thermostat is switched off when the guard $T \geq 9$ is enabled. Hence the inherent non-determinism in a Hybrid Systems specification: when in Cool, the system can jump to Heat whenever the temperature T is in the interval $[5, 6]$.

Hybrid dynamical systems provide a powerful framework for modeling and analyzing the complex and often unpredictable behavior of systems that exhibit both continuous and discrete dynamics, making them essential in various fields of engineering and science.

Chapter 3

An in-depth analysis of related works

In this chapter, we will analyze in detail the two primary papers used as inspiration and a starting point for the new approach that will be proposed in the next chapter.

3.1 Tron-Vidal approach

We will refer to the paper called "Distributed image-based 3D localization of Camera Sensor Networks" [7].
The **problem** addressed is the distributed estimation of N camera poses in CSN, using image measurements only.
The **goal** is to localise the network, i.e. to find cameras' relative poses, up to a global reference frame and scale factor from image measurements. In order to do that a suitable cost function is defined and it is shown how it an be minimized on $\mathbb{SE}(3)^N$. This space can be decomposed as $\mathbb{SO}(3) \times \mathbb{R}^3$, where the first space is used to estimate the attitude of the cameras and the second one to estimate the position.
Few **assumptions** are made in order to resolve the problem. We have:

- Low power sensor nodes equipped with a camera, able to communicate with each other;
- Each camera can extract a set of 2D points from each camera, neighboring cameras can match this 2D point to estimate the relative rotation and the direction of translation (two-view vision techniques);
- The scene is static (or equivalently all cameras are synchronized);
- The communication is lossless.

We analyze now the **method** used to obtain the pose estimation of the cameras. Firstly, the feature points extracted from the images obtained from each camera i and j are used to obtain a noisy estimate $(\tilde{R}_{ij}, \tilde{t}_{ij})$. Each camera can extract P_i feature points $x_i^{(p_i)}$, where $p_i = 1...P_i$, in homogeneous coordinates, *i.e.* they are points in $\mathbb{P}^2$ and correspond to the projection of points in 3D space. Then it is assumed that there is correspondence in each edge $(i,j) \in \mathcal{E}$ between points in images i and j that can be established, namely $x_i^{(p_i)^T} \hat{T}_{ij} R_{ij} x_j^{(p_j)} = 0$. Given enough points, the epipolar constraint estimates $E = \hat{T}_{ij} R_{ij}$ allows to define $\tilde{t}_{ij} = \frac{\tilde{T}_{ij}}{||\tilde{T}_{ij}||}$ and $\tilde{R}_{ij}$ up to a scale factor on T_{ij}. The goal now is to find a set of relative transformations g_{ij} such that the localised network is as close as possible to the relative measurements $\tilde{g}_{ij}$, where $g_{ij} = g_i^{-1} \odot g_j$.
This can be done using the least squares, where the cost function is defined as

$$\varphi = \frac{1}{2} \sum_{i \to j} d_g^2(g_{ij}, \tilde{g}_{ij}) = \frac{1}{2} \sum_{i \to j} \left(d^2_{\mathbb{SO}(3)}(R_{ij}, \tilde{R}_{ij}) + ||T_{ij} - \tilde{T}_{ij}||^2 \right) \tag{3.1}$$

To minimize the function in a distributed way and keep the network localised there is the need to reparametrize every transformation g_{ij} with absolute transformations g_i and g_j.

$$\varphi(\{R_i\}, \{T_i\}, \{\lambda_{ij}\}) = \frac{1}{2} \sum_{i \to j} d_g^2(g_i^{-1} g_j, \tilde{g}_{ij}) \tag{3.2}$$

$$= \frac{1}{2} \sum_{i \to j} \left(d^2_{\mathbb{SO}(3)}(R_i^T R_j, \tilde{R}_{ij}) + ||R_i^T (T_j - T_i) - \lambda_{ij} \tilde{t}_{ij}||^2 \right) \tag{3.3}$$

$$= \frac{1}{2} \sum_{i \to j} \left(\varphi_R(\{R_i\}) + \varphi_T(\{R_i\}, \{T_i\}, \{\lambda_{ij}\}) \right) \tag{3.4}$$

The complete cost function φ has multiple local minima if the starting point is a random configuration.
Regarding the estimation of the **rotations**, the consensus framework with Riemannian gradient descent is used. R_i is the rotational part of the absolute pose g_i, each node k compute the gradient of φ_R with respect to its rotation R_k.
For the estimation of the **translation**, the approach is similar to the one used above, hence the goal is to minimize φ_T subject to $\lambda_{ij} \geq 1$ using gradient descent with respect to both T_k and λ_{lk}.
The right choice of the **step size** is fundamental in order to minimize φ_T and can be chosen by considering the degrees of the nodes

$$\mathcal{E} \in \left(0, 2(\max\{9, 4\Delta_G\})^{-1} \right) \tag{3.5}$$

3.2 Dual quaternion approach

We will refer to the paper called "Distributed dual quaternion based localization of visual sensor networks" (DDQL)[6].
The **problem** addressed here is the localization for a VSN, hence the self-estimation of the position and attitude of each visual sensor.
The **goal** here is to resolve the problem of the Tron-Vidal approach regarding the accumulation of the errors: since the orientation estimation in $\mathbb{SO}(3)$ is done first, the position estimates accumulate more errors. This means that the goodness of the final estimation strictly depends on the initial guess. This approach uses the unit dual quaternion to unitarly represent the camera pose and consequently redefine the least square cost function so that the estimation error is balanced and the algorithm is more robust.
Some **assumptions** are made also here. We have:

- Dual quaternion is used $d = q_r + \varepsilon q_d \in DH$ (double quaternion space) where $q_r, q_d \in H, \varepsilon^2 = 0, \varepsilon \neq 0$. A dual quaternion is called unit dual quaternion when $||q_r||^2 = 1, q_r^T q_d = 0$ so that $d \in DH_u$ (double unitary quaternion space) and can be used to describe a pose transformation in 3D space;
- VS of n cameras in which each device is associated with a local frame $\mathcal{F}_i = \{O_i, (X_i, Y_i, Z_i)\}$. The pose $g_i \in \mathbb{SE}(3)$ is identified by the position and orientation of $\mathcal{F}_i$ with respect to the global inertia frame;
- The network is modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
- Each camera uses standard computer vision algorithms to recover a noisy measurement $\tilde{g}_{ij} \in \mathbb{SE}(3)$ of the relative pose with respect to any device in its neighborhood;
- The body frame of camera 1 coincides with the global reference frame d_i.

They want to compute the set of relative transformations $\hat{g}_{ij}$, that satisfy the consistency constraint along network cycles and are as close as possible to the available noisy measurements $\tilde{g}_{ij}$.
The **method** used here is to define a least squares cost function

$$\rho(\{\hat{g}_{ij}\}) = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \frac{1}{2} d_g^2(\hat{g}_{ij}, \tilde{g}_{ij}) \tag{3.6}$$

Hence,

$$\rho(\{\hat{g}_i\}) = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \frac{1}{4} \left(d_g^2(\hat{g}_i^{-1} \odot \hat{g}_j, \tilde{g}_{ij}) + d_g^2(\hat{g}_j^{-1} \odot \hat{g}_i, \tilde{g}_{ji}) \right) \tag{3.7}$$

$$= \sum_{i \in \mathcal{V}} \rho_i(\hat{g}_i) \tag{3.8}$$

This cost function can be reformulated using dual quaternions

$$\rho(\{\hat{d}_i\}) = \sum_{j \in \mathcal{N}_i} \frac{1}{4} \left(||U(\tilde{d}^{\star}_{ij})\tilde{V}(\hat{d}_j)\hat{d}_i||^2 + ||U(\tilde{d}^{\star}_{ji})\tilde{V}(\hat{d}_i)\hat{d}_j||^2 \right) \tag{3.9}$$

$$= \sum_{i \in \mathcal{V}} \rho_i(\hat{d}_i) \tag{3.10}$$

The minimization of $\rho(\{\hat{d}_i\})$ can be performed in a distributed fashion with each device locally exploiting an iterative optimization strategy:
After the initialization step:

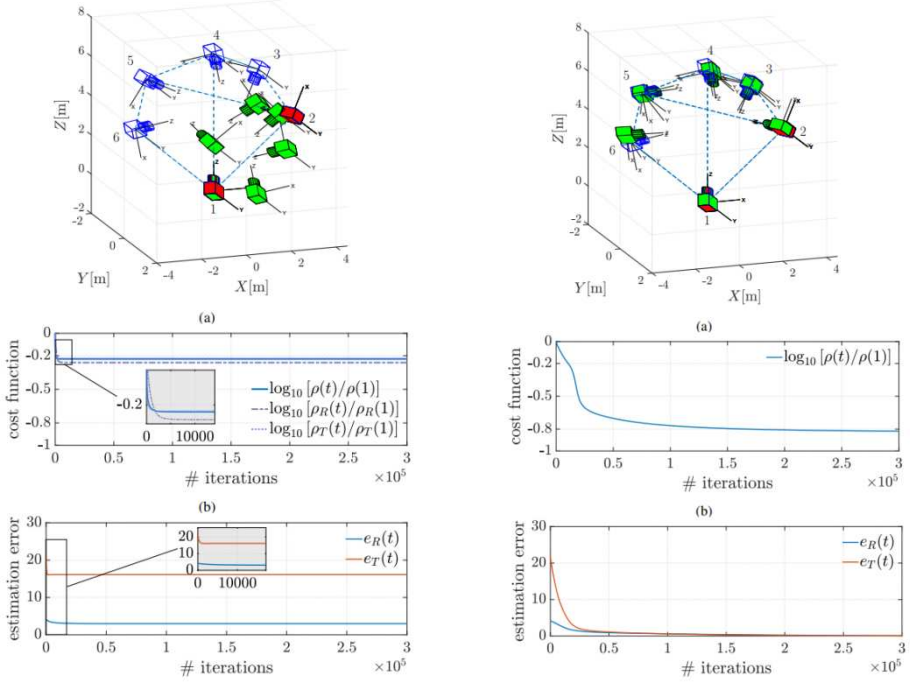
1. Determine the derivative of $\rho_i(t)$ with respect to $\hat{d}_i(t)$;
2. Determine the estimation $\hat{d}_i(t+1)$ using steepest gradient descent;
3. Compute a normalization step since $\hat{d}_i(t+1)$ may violate the constraints of DH_u, the pose estimate $\hat{d}'_i(t+1)$ is then communicated to all the neighboring nodes $j \in \mathcal{N}_i$

3.3 Main differences between TV and DDQL

It is easy to notice that the biggest differences are the spaces over which the estimation is done and the divided/united estimation of the pose of each camera. The TV algorithm finds a good solution only if the initial guess is close enough to the real cameras' poses and the error is not evenly distributed between rotation and translation estimation since the rotation step is done before and the translation error is bigger (accumulation of rotation error + translation error). The dual quaternion approach is superior to the one above since it is able to converge to a good solution, despite the goodness of the initial guess with a small error.

In Fig. 3.1 we can see the performance comparison of the two algorithms. We have the TV one in Fig. 3.1a, and the DDQL in Fig. 3.1b.
In both cases, the setup is noise-free meaning that relative and real poses correspond and the network has 6 cameras. As we can clearly see the initial estimates (in red) are really far from the real camera poses (in blue). Moreover, the first camera pose is initialized with its true one, otherwise, the estimation will not be possible, and all the other pose estimates are initialized as one of the other cameras' true poses (camera 2 in the example).

In the comparison figures the final estimates are represented in green and it is obvious that the TV algorithm is not able to obtain an accurate final estimate if the initial one is not good. The algorithm, after a few initial iterations, gets stuck in a local minimum. Lastly, as expected, the decrease in the rotation error is bigger than the one regarding the transition one.



(a) TV algorithm with bad initial starting conditions

(b) DDQL algorithm with bad initial starting conditions

Figure 3.1: Performances confront, from [6]

On the other hand, it is clear that the DDQL algorithm is more robust with respect to initial conditions, in fact, the final estimated poses are very close to the real ones, the cost function decreases without getting stuck in a local minimum and the errors (rotational and translational) tends to zero.

3.4 New contribution

The idea, which we introduce now but will be explained in the next chapter, is to find a better solution to the TV algorithm that does not depend on the initialization configuration and possibly has an evenly distributed error between the rotation and translation steps.
To do so, instead of using space separation or double quaternion, we will exploit the simple unit quaternion q, remembering that there is the 'double coverage' property.

Chapter 4

Distributed Hybrid Unit Quaternion Localisation of CSN

4.1 The idea

The problem we address is the self-estimation of the position for a Visual Sensor Network, made up of N cameras, using image measurements only.
The goal is to localise the network using the unit quaternion, hence the space where the optimization takes place is the $3-n$ sphere $\mathbb{S}^3$. The idea is to optimize (on the manifold) using the simple unit quaternion as the main object to define a suitable cost function and minimize it.

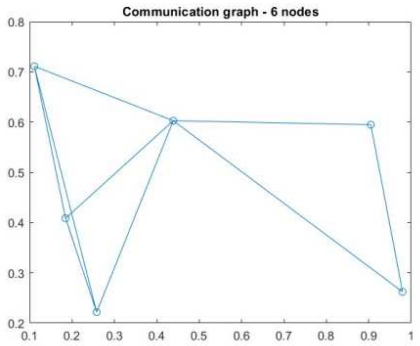
We want to solve the big problem that arises with the TV approach, hence the fact that the optimization goodness depends on the goodness of the initialization. To do so we decide to use the unit quaternion, and not double one, because we want to exploit the double coverage property using a hybrid law.
The idea is that, since $R(q) = R(-q)$ we could, at random points, jump from q_i to $-q_i$ to try to obtain a faster convergence to an almost zero error solution.

We suppose that:

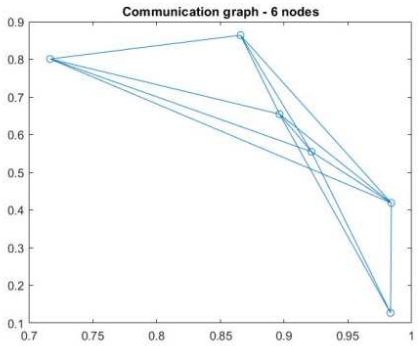
- Unit quaternion is used $q \in H$ to describe the pose transformation in the 3D space;
- All cameras are synchronized;
- The network is modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
- Each camera can extract a set of 2D points from the scene and, using standard computer vision algorithms, is able to recover a noisy measurement $\tilde{g}_{ij} \in \mathbb{SE}(3)$ of the relative pose with respect to any device in its neighborhood;

- The position of camera 1 is fixed and coincides with the global reference frame ($q_1 = q_I$).

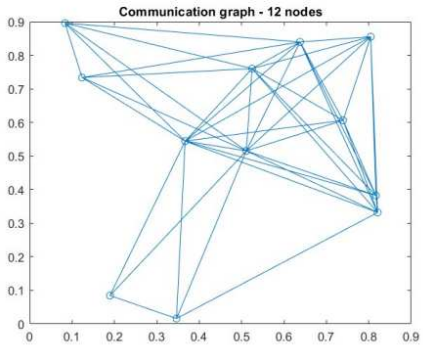
The method used here is to generate a random network with n nodes, that is not fully connected in order to have afterward a distributed algorithm, similar to what obtained in Fig. 4.1.
From the adjacency matrix of the graph (using computer vision techniques), we are able to retrieve all the noisy measurements $\tilde{g}_{ij} \in \mathbb{SE}(3)$ of the network.
After, we define a suitable cost function using quaternions and we try to minimize it.



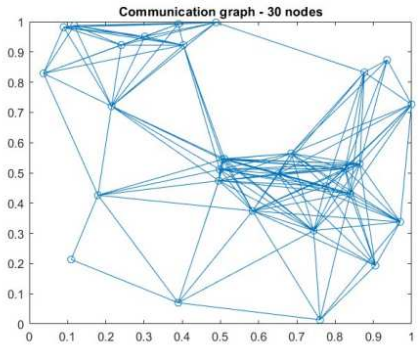
(a) Possible configuration with 6 nodes



(b) Another possible configuration with 6 nodes



(c) Possible configuration with 12 nodes



(d) Possible configuration with 30 nodes

Figure 4.1: Possible configuration with 6 nodes

4.2 The setup

We define the following quaternions, see Fig.(4.2):

1. $q_i \rightarrow$ Real quaternion (unknown)
2. $\hat{q}_i \rightarrow$ Local estimates of the real quaternions (associated with each node)
3. $\tilde{q}_{ij} \rightarrow$ Relative noisy measurements of $q_j \odot q_i^{-1}$ available to nodes i and j

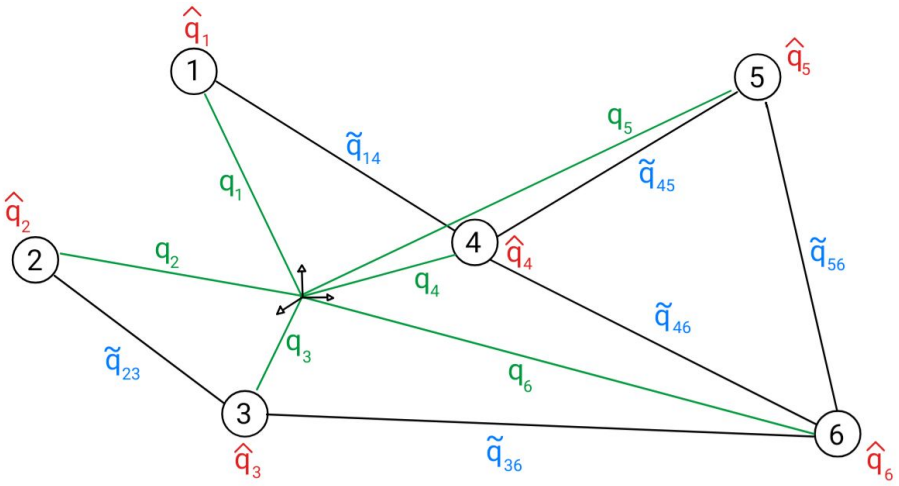


Figure 4.2: Quaternions description on the camera setup

We use a graph that is undirected for the representation, namely:

$$(i,j) \in \mathcal{E} \iff (j,i) \in \mathcal{E}$$

Notice that $\tilde{q}_{ij}$ is different from $\tilde{q}_{ji}$, due to the fact that $q_1 \odot q_2 \neq q_2 \odot q_1$ and the noise on the measurement.

4.3 Cost function calculation

Consider measurements $\tilde{q}_{ij}$ of $q_j \odot q_i^{-1}$ for each $(i,j) \in \mathcal{E}$, that are fixed. Then, following what has been introduced previously in 2.3.1, we introduce the reasonable cost

$$J = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} d(\tilde{q}_{ij}, \hat{q}_j \odot \hat{q}_i^{-1}) \tag{4.1}$$

$$= \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} d(\tilde{q}_{ij}^{-1}, \hat{q}_i \odot \hat{q}_j^{-1}) \tag{4.2}$$

$$= \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} |\delta_{ij}|^2 \tag{4.3}$$

Where we may express

$$\delta_{ij} = \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ij}^{-1})(\hat{q}_j \odot \hat{q}_i^{-1}) = \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ij}^{-1}) G(\hat{q}_i^{-1}) \hat{q}_j \tag{4.4}$$

or equivalently

$$\delta_{ij} = \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ij})(\hat{q}_i \odot \hat{q}_j^{-1}) = \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ij}) G(\hat{q}_j^{-1}) \hat{q}_i \tag{4.5}$$

We consider then the derivative of our cost function J, namely $\dot{J}$ that is defined as

$$\dot{J} = \sum_i (\nabla_i J(q))^T \dot{q}_i \tag{4.6}$$

Where $\nabla_i J(q) \in \mathbb{R}^4$ denotes the gradient of the cost with respect to q_i.
We can define the vector $q = (\hat{q}_1, \hat{q}_2, ..., \hat{q}_N)$.
Using the expression above we may compute the gradient $\nabla_i J(q)$ of J with respect to a generic q_k as follows:

$$(\nabla_k J(q))^T = \sum_{(k,j)\in\mathcal{E}} \delta_{kj}^T \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{kj}) G(\hat{q}_j^{-1}) + \sum_{(i,k)\in\mathcal{E}} \delta_{ik}^T \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ik}^{-1}) G(\hat{q}_i^{-1}) \tag{4.7}$$

$$= \sum_{(k,j)\in\mathcal{E}} \hat{q}_k^T G(\hat{q}_j) F(\tilde{q}_{kj}^{-1}) \begin{bmatrix} \mathbf{0}^T \\ \mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{kj}) G(\hat{q}_j^{-1}) \tag{4.8}$$

$$+ \sum_{(i,k)\in\mathcal{E}} \hat{q}_k^T G(\hat{q}_i) F(\tilde{q}_{ik}) \begin{bmatrix} \mathbf{0}^T \\ \mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ik}^{-1}) G(\hat{q}_i^{-1})$$

$$= \sum_{(k,j)\in\mathcal{E}} \hat{q}_j^T G(\hat{q}_k) F(\tilde{q}_{kj}) \begin{bmatrix} \mathbf{0}^T \\ \mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{kj}) G(\hat{q}_j^{-1}) \tag{4.9}$$

$$+ \sum_{(i,k)\in\mathcal{E}} \hat{q}_i^T G(\hat{q}_k) F(\tilde{q}_{ik}^{-1}) \begin{bmatrix} \mathbf{0}^T \\ \mathrm{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathrm{I}_3 \end{bmatrix} F(\tilde{q}_{ik}^{-1}) G(\hat{q}_i^{-1})$$

That is a row vector of dimension 1×4, hence $\nabla_k J(q)$ is a column vector of dimension 4×1.

And finally, the derivative of the cost function becomes:

$$\dot{J} = \sum_{k\in\mathcal{N}} (\nabla_k J(q))^T \dot{\hat{q}}_k \tag{4.10}$$

$$= \sum_{k\in\mathcal{N}} (\nabla_k J(q))^T \frac{1}{2} F_2(\hat{q}_k)\omega_k \tag{4.11}$$

$$= \sum_{k\in\mathcal{N}} \frac{1}{2} \underbrace{\nabla_k J(q))^T}_{1\times 4} \underbrace{\begin{bmatrix} -\hat{\boldsymbol{\epsilon}}_k^T \\ \hat{\eta}_k \mathrm{I}_3 + \boldsymbol{S}(\hat{\boldsymbol{\epsilon}}_k) \end{bmatrix}}_{4\times 3} \omega_k \tag{4.12}$$

$$\overset{\text{symbolically}}{=} \sum_{k\in\mathcal{N}} \frac{1}{2} \underbrace{\begin{bmatrix} * & * & * \end{bmatrix}}_{1\times 3} \omega_k \tag{4.13}$$

To have negative semi-definiteness:

$$\omega_k = -\mu \left((\nabla_k J(q))^T F_2(\hat{q}_k) \right)^T \tag{4.14}$$

$$= -\mu F_2(\hat{q}_k)^T \nabla_k J(q) \tag{4.15}$$

$$= -\mu \underbrace{\begin{bmatrix} -\hat{\boldsymbol{\epsilon}}_k & \hat{\eta}_k \mathrm{I}_3 + \boldsymbol{S}(\hat{\boldsymbol{\epsilon}}_k) \end{bmatrix}}_{3\times 4} \underbrace{\nabla_k J(q)}_{4\times 1} \tag{4.16}$$

being μ an arbitrary gain.
We can finally rewrite $\dot{J}$

$$\dot{J} = -\sum_k \frac{1}{2} \mu (\nabla_k J(q))^T F_2(\hat{q}_k) F_2(\hat{q}_k)^T \nabla_k J \tag{4.17}$$

4.4 Matlab implementation

Regarding the Matlab implementation, as stated above, we start with the creation of a random graph, not fully connected. On this graph, each node represents a camera, defined by a quaternion. From each camera, we are able to retrieve noisy measurements $\tilde{q}_{ij}$. These measurements are corrupted by a white Gaussian noise that has a standard deviation of $5deg$.

We have then to initialize the optimization algorithm. This can be either done by initializing all the cameras at q_I, so at the identity quaternion, or by initializing them in a neighborhood of the final solution (as done in the TV approach).
In Fig.4.3 it is shown the continuous-time implementation in Simulink.

The minimization of the cost function 4.3 is performed following the distributed formalism. In detail, each device in the network is able to locally exploit

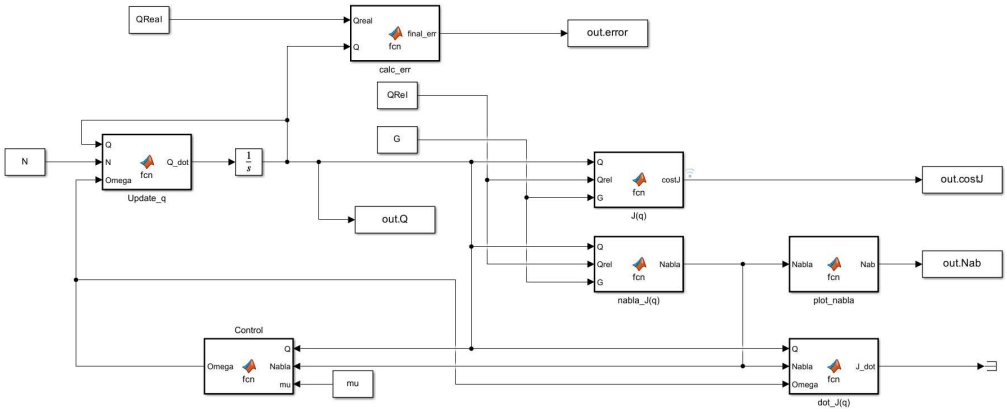


Figure 4.3: Simulink implementation scheme

an iterative optimization strategy to minimize the cost. For the implementation, at every timestamp t each node i computes the following procedure:

- After receiving the relative measurements, we initialize the quaternions estimate $\hat{q}_i = q_I$;
- We calculate the cost $J(t)$4.3, the gradient of the cost $\nabla_i J(t)$4.7, the derivative of the cost function $\dot{J}(t)$4.11 and $\omega(t)$4.14;
- We update $\dot{q}_i(t) = \frac{1}{2} F_2(q_i(t))\omega_i(t)$2.15;
- We perform a normalization step on $\dot{q}_i(t+1)$;
- We integrate, finding $\hat{q}_i(t+1)$. This new estimate will be communicated to all the neighboring nodes $j \in \mathcal{N}_i$.

Algorithm 1 Distributed Unit Quaternion

$//t_{max}$ time length of the simulation
$//n$ number of cameras in the network
Input: relative noisy measurements $\{\tilde{q}_{ij} \in \mathcal{E}\}$
Initialization: (t=0):
initial pose estimates $\{\hat{q}_i(0), i \in \mathcal{V}\}$, $\hat{q_1}(0) = q_I$
for $t = 1 : t_{max}$ **do**
  $\hat{q_1}(0) = q_I$
  for $i = 2 : n$ **do**
    Calculate: $J(t)$, $\nabla_i J(t)$, $\dot{J}(t)$, ω_i
    Update: $\dot{q}_i(t) = \frac{1}{2} F_2(q_i(t))\omega_i(t)$
    Normalization: of $\dot{q}_i(t+1)$
    Integration: to find $\hat{q}_i(t+1)$
  end for
end for

The algorithm stops after an amount of time $t_{max} \in \mathbb{N}$, large enough to guarantee the achievement of a minimum for the cost function.

4.4.1 Hybrid approach

As stated above we would like to introduce a hybrid law to make the convergence to the desired solution faster. This involves leveraging the dual coverage characteristic of quaternions, allowing a suitable quaternion jump at each instance.

The method is the following:

- We compare the value of the cost function at t and at $t-1$, if $J(t-1)-J(t) < \delta$ we will "jump", where δ is the threshold. This threshold has to be chosen small enough so that jumps are recurrent, but at the same time big enough such that we do not have a jump at every iteration;
  $\rightarrow$ So, if the cost function is not decreasing enough, we can apply a hybrid law.

- We change the value of one quaternion at a time from q_i to $-q_i$ and we calculate the new value of the cost function;

- We confront the various cost functions and we keep the one with the smallest value, meaning that the difference between the value of the cost function at the previous instance and the one that we are keeping will be the biggest possible;

- If the transition leads to a significant decrease in the cost function, the quaternion that corresponds to the changed cost function at $-q$, will be kept to $-q$, while all the others are back to their original value.

Thus, if the transition leads to a significant decrease in the cost function, the quaternion q will jump to $-q$ to achieve accelerated convergence.
In this way, we should be able to have an algorithm that performs much better than the original one.

What we saw from the simulations is that there are no jumps at all, even when the threshold is set to be very small. This happens because the cost function is symmetric, hence every sign change will be canceled.

A solution could be to change the norm of the cost function or to directly change the function itself, in order to have something not symmetric. We tried to keep the cost function as it is and to use a different hybrid approach.

Algorithm 2 Hybrid approach #1

```
//n number of cameras in the network
//δ improvement threshold
//J(t) value of cost function at iteration t
//J(t − 1) value of cost function at iteration t − 1
if J(t − 1) − J(t) < δ then
    for i = 1 : n do
        Change the value of q̂_i to −q̂_i
        Calculate: J(t)'_{q_i}
        Change the value of −q̂_i back to q̂_i
    end for
    Compare all new values of cost function J(t)'_{q_i}, keep the smallest one
    if J(t − 1) − J(t)'_{q_i} < δ then
        Switch the value of q̂_i with −q̂_i
    else
        Do not change any quaternion to the corresponding negative
    end if
end if
```

The second idea is to find bad enough initial conditions, so as to have a very slow decrease in the cost function or possibly get stuck in a local minimum. In order to do so, we use the function *fmincon()* in Matlab and as the objective function we use

$$\text{obj} = \beta \left(J(q_0) - J(q_f) \right) - \alpha J(q_f) \tag{4.18}$$

where α and β are arbitrary coefficients, and α is possibly bigger than β so that the fact that the final estimation of q_f needs to be bad is stressed. What we are trying to look for is an initial condition, so values for q_0, that do not enable the cost function do decrease significantly. Hence we will have a final condition on q, called q_f, not close to the real one. The error in the estimation will be big and we will have to resume to hybrid techniques to make the converge faster.

The idea here is the following:

- Use *fmincon()* to find bad initial conditions for the quaternions;
- For every iteration randomly choose if we want to perform the hybrid optimization (i.e. is done with a probability of 20%);
- If we decide to do it, we need to choose (could be done randomly) a camera i, hence a quaternion q_i, of which the value will be changed;
- Using the adjacency matrix, we check to which cameras i is connected and we choose (again, this could be done randomly) a camera j, hence a quaternion q_j so that $\tilde{q_{ij}}$ exists;

- We define a new value for q_i as follows

$$\hat{q}_{i_{new}} = \tilde{q}_{ij}^{-1} \odot \hat{q}_j \tag{4.19}$$

Thus, if we have a slow convergence we can try to make it faster by changing the value of a quaternion q_i at a time.

Algorithm 3 Hybrid approach #2

$//n$ number of cameras in the network
$//q_0$ initial set of conditions for the quaternions
$//\sigma$ probability of applying hybrid law (i.e. if we want to apply it with 20% probability, $\sigma = 80\%$)
Initialize: $fmincon()$
$\rightarrow$ **Constraints:** $-1 \leq q_{0_{jk}} \leq, \forall j, k$ and $\left\|\hat{q}_{0_i}\right\|^2 = 1$
$\rightarrow$ **Objective function:** $\beta\left(J(q_0) - J(q_f)\right) - \alpha J(q_f)$
Generate a random number x (between 1-100)
if $x > \sigma$ we apply hybrid law **then**
    Choose at random a quaternion $\hat{q}_i,\ i = 2...n$ to change (q_1 is fixed and do not change)
    Find the quaternions connected to the one we want to change using the adjacency matrix
    Choose at random one of the quaternions connected to $\hat{q}_i$, say $\hat{q}_j$
    Calculate: $\hat{q}_{i_{new}} = \tilde{q}_{ij}^{-1} \odot \hat{q}_j$
end if

4.5 Numerical results

We will show in this section the results obtained using the techniques described above.
We will consider only the rotation estimation therefore for the representation of the 3D space we will place the cameras at the same height and in the positions retrieved from the communication graphs, such as the ones in Fig. 4.1. Networks of different sizes (*i.e.* with $n = 6,\ n = 12$ and $n = 30$) will be considered to show the effectiveness of our approach.

In every simulation, we have that the blue cameras are the real cameras' poses, the red ones represent the initial position, notice that camera 1 is initialized with its true position, and the green ones represent the final estimate.

The estimation error is calculated by summing all the single errors on the quaternions

$$e_q(t) = \sum_{i \in \mathcal{V}} \left\|q_i - \hat{q}_i(t)\right\|^2 \tag{4.20}$$

4.5.1 Complete simulations of networks of different sizes

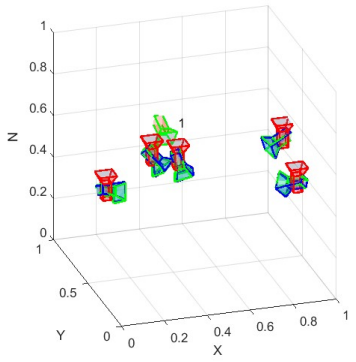


Figure 4.4: 3D of a 6 a-camera network

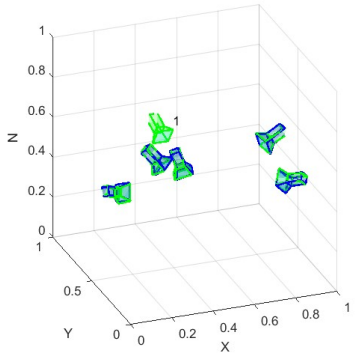


Figure 4.5: 3D representation of a 6-camera network, without cameras representing the initial estimate

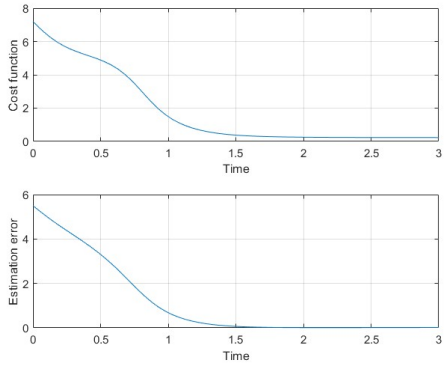


Figure 4.6: Cost and error functions, of a 6 a-camera network

As we can see, for a small network made by $n = 6$ cameras, the error on the quaternions tends to zero and the cost function is almost reaching zero too.

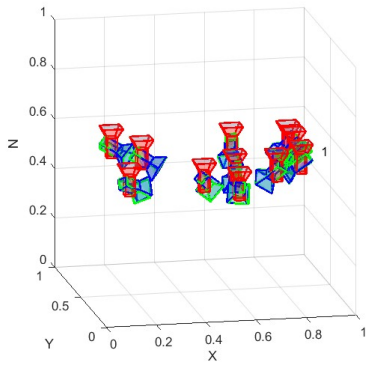


Figure 4.7: 3D representation of a 12-camera network

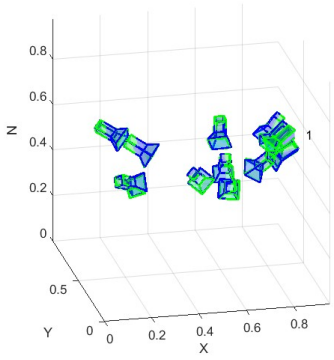


Figure 4.8: 3D representation of a 12-camera network, without cameras representing the initial estimate

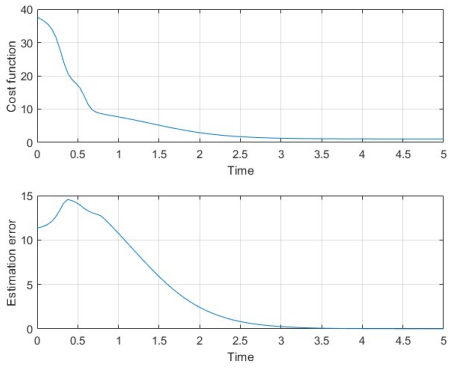


Figure 4.9: Cost and error functions, of a 12-camera network

The same reasoning can be done for a slightly bigger network made by $n = 12$ cameras, where the error on the quaternions tends to zero and the cost function is almost reaching zero too.

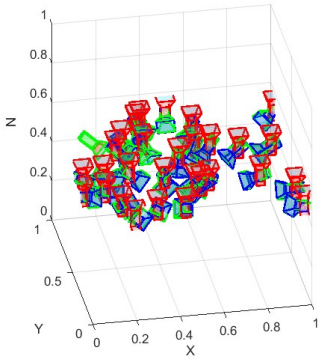


Figure 4.10: 3D representation of a 30-camera network

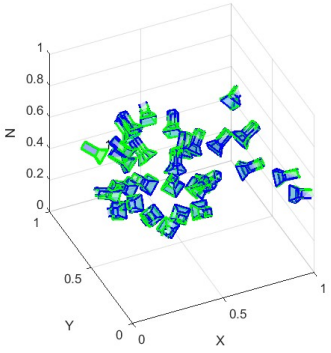


Figure 4.11: 3D representation of a 30-camera network, without cameras representing the initial estimate

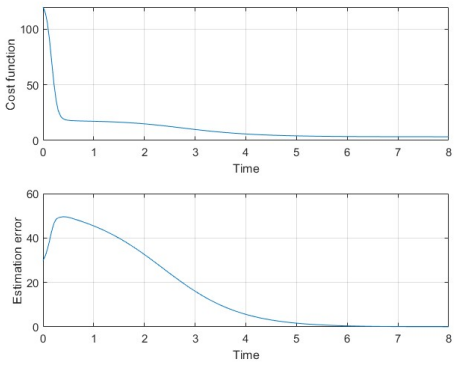


Figure 4.12: Cost and error functions, of a 30-camera network

Again, it can be done for a big network made by $n = 30$ cameras, where the error on the quaternions tends to zero and the cost function is almost reaching zero too.

4.5.2 Comparison with a setup where different values of σ on $\hat{q}_1$ are used

We consider different-sized networks and an ideal setup where the relative measurement $\tilde{q}_{ij}$ is exact (*i.e.* the noise variance on this measure is zero).

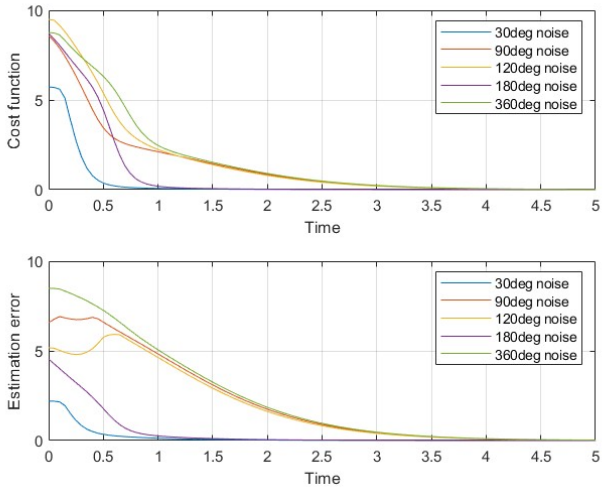


Figure 4.13: Comparison of different values of σ on initial estimate ($n = 6$)

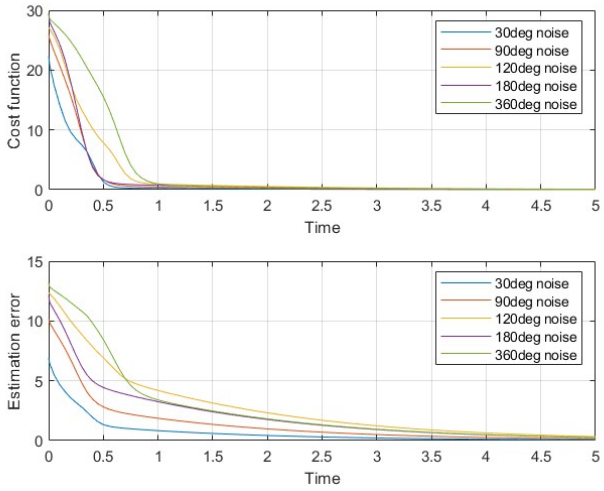


Figure 4.14: Comparison of different values of σ on initial estimate ($n = 12$)

In order to prove robustness with respect to the initial estimate on the quaternions, we illustrate the results obtained with various initial conditions.

These initial conditions are obtained by perturbing the quaternion representing the real pose with a white Gaussian noise of varying standard deviation σ.

For each network we use the following values of σ: $30deg$, $90deg$, $120deg$, $180deg$, $360deg$.

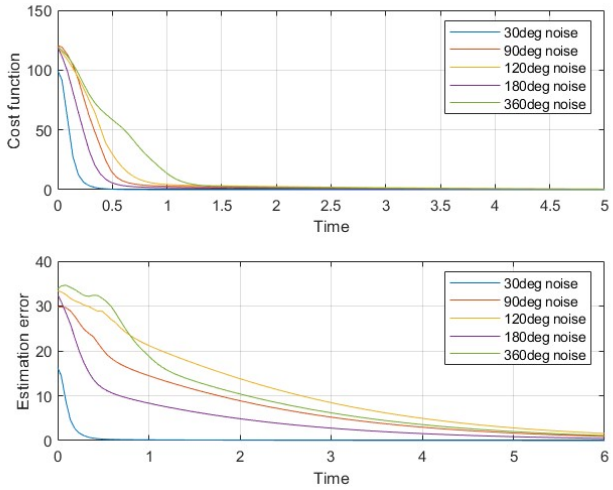


Figure 4.15: Comparison of different values of σ on initial estimate ($n = 30$)

From all the simulations it is clear that, in every network considered and for every initial condition both the values of cost function and error on quaternion estimation converge toward zero.

We can say that we have strong robustness with respect to initial conditions values.

4.5.3 Comparison with a setup where different values of σ on $\tilde{q}_{ij}$ are used

Here we will show what happens if we consider different values for the white Gaussian noise standard deviation parameter, used to corrupt the relative measure $\tilde{q}_{ij}$.

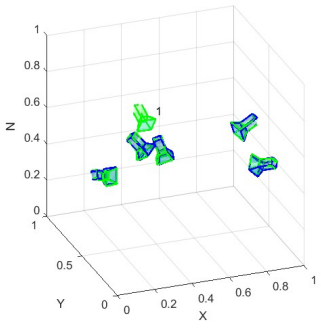


Figure 4.16: Network representation with $\sigma = 5deg$

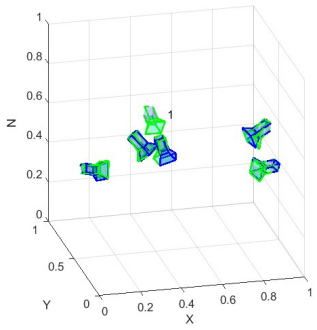


Figure 4.17: Network representation with $\sigma = 10deg$

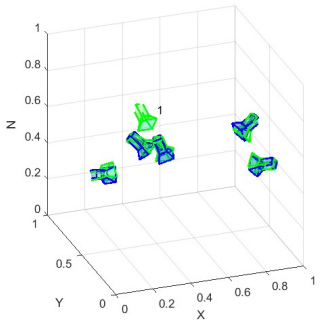


Figure 4.18: Network representation with $\sigma = 15deg$

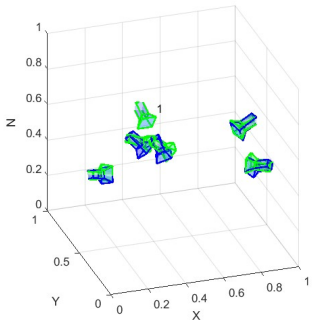


Figure 4.19: Network representation with $\sigma = 20deg$

In the pictures above we can see the 3D representation of the same network composed of 6 cameras, where we omit the plot of the initial estimate in order to have a clearer view of the cameras (we are using the same setting and the same network of Fig. 4.4).
It is clear that, as expected, the more variance we add, the less accurate the final estimation of the camera rotation will be.

In Fig. 4.20 we have the comparison of cost function J and estimation error on the quaternion e_q for all the different values of σ. We start from the ideal case

where we have no corruption at all, $\sigma = 0$, and, incrementing by a value of 5 in each simulation, we arrive at $\sigma = 20$. There is a need to take into consideration that, for precision application, such a big variance will never appear, but it is still interesting to see the behavior of the algorithm with this value.

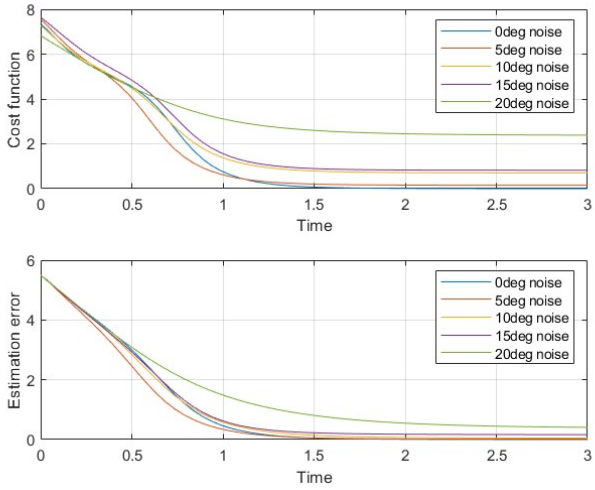


Figure 4.20: Comparison of different values of σ ($n = 6$)

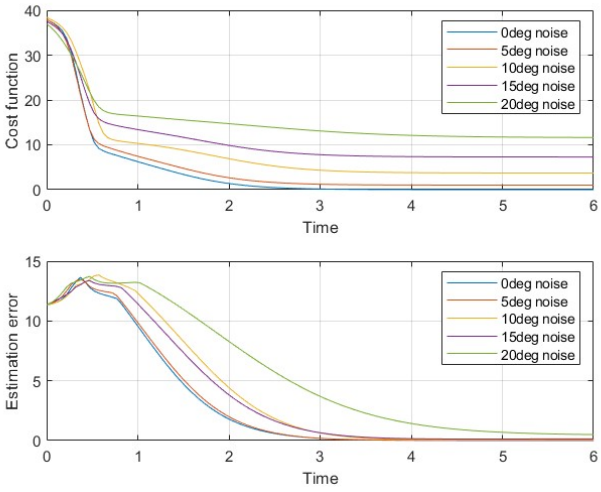


Figure 4.21: Comparison of different values of σ ($n = 12$)

For the small network, we can see that with the very realistic parameter $\sigma = 5$ we have the convergence of J to a value very close to zero.

Regarding what is obtained with $\sigma = 10$ and $\sigma = 15$ we have very comparable results. In fact, we have that for both values the cost function reaches a steady state value close to one.
As expected, with a standard deviation value too big we have very bad performances. This is visible from both the cost function plot and the 3D network representation of Fig. 4.19.

In Fig. 4.21 we do the same comparison, with a network made by 12 cameras. The increasing size of the network implies the growing difficulty in the estimation.
Similarly to the 6 cameras network, with a small deviation $\sigma = 5$, we obtain very good results. For all the other values of standard deviation, we have convergence towards big values for the cost function.

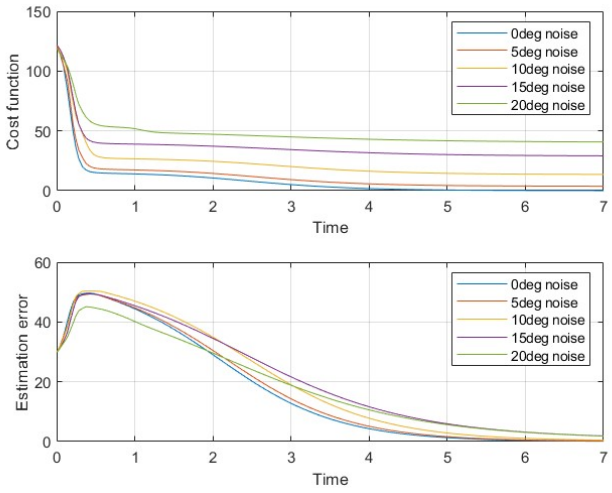


Figure 4.22: Comparison of different values of σ ($n = 30$)

In Fig. 4.22 we analyze a very big network made by 30 cameras.
The results are similar to the ones obtained for the 12 cameras network, with a small steady-state value when $\sigma = 5$, while we have convergence towards big values for all the other values.

4.5.4 Hybrid algorithm #1

We have already introduced above the reason why the first hybrid algorithm is not working, but in this section, we will present a detailed analysis of the problem of this approach.

We start by plotting the cost function in the cases where do not or do apply the hybrid algorithm 2.
The two plots in Fig. 4.23 are overlapping; this happens because the hybrid approach produces no jumps at all, despite the value of δ used in
$J(t-1) - J(t) < \delta$.

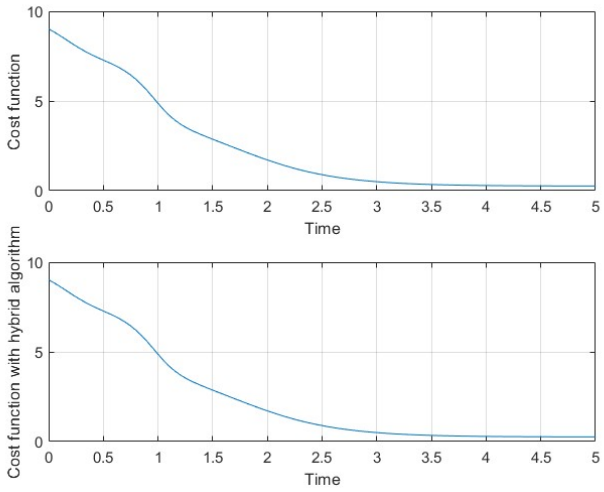


Figure 4.23: Comparison of the cost function with and without the use of the hybrid algorithm

To have a better understanding of what exactly is happening we consider a network made by 6 cameras. It is fundamental to have the adjacency matrix that describes the network connections, hence

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{4.21}$$

It is useful to recall the explicit formulation of δ_{ij}

$$\begin{aligned}\delta_{ij} &= \begin{bmatrix}\mathbf{0} & \mathrm{I}_3\end{bmatrix} F(\tilde{q}_{ij})G(\hat{q}_j^{-1})\hat{q}_i \\ &= \begin{bmatrix}a\\b\\c\end{bmatrix}\end{aligned}$$

and of the cost function

$$\begin{aligned}J &= \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} |\delta_{ij}|^2 \\ &= \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \begin{bmatrix}a & b & c\end{bmatrix}\begin{bmatrix}a\\b\\c\end{bmatrix}\end{aligned}$$

We can analyze what happens when we jump from q to $-q$, step by step. If we look at 4.22 we can see that are highlighted all the connections of node 2, hence all the values of δ_{ij} that will be affected from a jump of q_2 to $-q_2$.

$$A_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{4.22}$$

More in detail what happens is that the only thing changing when we jump is, of course, the sign of the quaternions that jump (one jump at a time). Looking at the effect of a jump of q_2, we have that the value of δ_{ij} becomes

$$\begin{aligned}\delta_{2j} &= \begin{bmatrix}\mathbf{0} & \mathrm{I}_3\end{bmatrix} F(\tilde{q}_{2j})G(\hat{q}_j^{-1})\hat{q}_2 \\ &= \begin{bmatrix}-a\\-b\\-c\end{bmatrix}\end{aligned}$$

and, for symmetry

$$\begin{aligned}\delta_{i2} &= \begin{bmatrix}\mathbf{0} & \mathrm{I}_3\end{bmatrix} F(\tilde{q}_{i2})G(\hat{q}_2^{-1})\hat{q}_i \\ &= \begin{bmatrix}-a\\-b\\-c\end{bmatrix}\end{aligned}$$

We have that the value of $F(\tilde{q}_{ij})$ is fixed since it is a measure; then there is one change of sign in either $G(\hat{q}_j^{-1})$ or in $\hat{q}_i$.
It is obvious that, being the sign of δ_{ij} the only change, this will be canceled when we calculate the cost function by multiplying δ for itself. This multiplication makes the jump completely useless.

The same exact reasoning can be done for the jumps of all the other nodes.

4.6 Comparison with Tron-Vidal algorithm

We can compare the rotation estimation of the TV algorithm and of our unit quaternion approach.

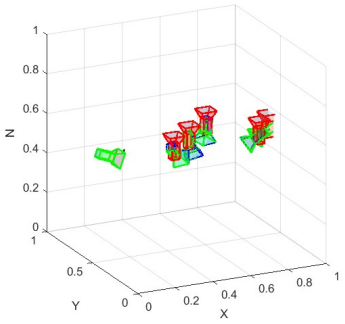


Figure 4.24: Network representation with the use of quaternions

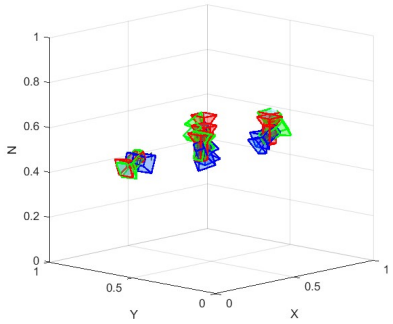


Figure 4.25: Network representation obtained with TV

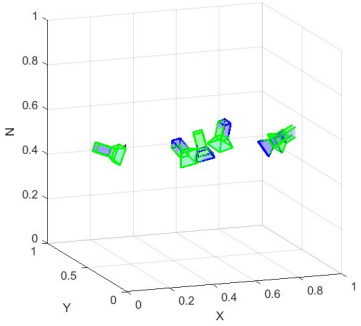


Figure 4.26: Network representation with the use of quaternions, without cameras representing the initial estimate

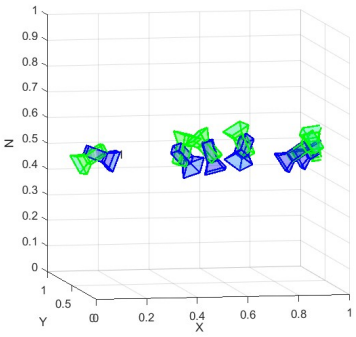


Figure 4.27: Network representation obtained with TV, without cameras representing the initial estimate

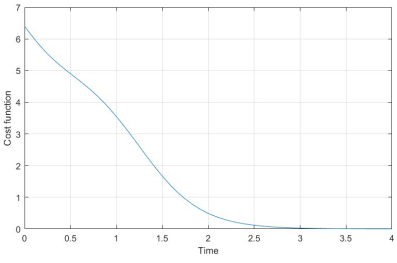


Figure 4.28: Cost function, using quaternions

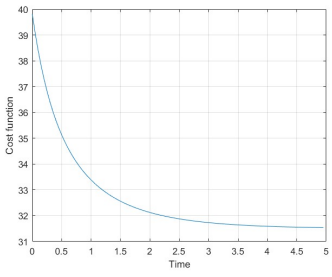


Figure 4.29: Cost function, using TV

To obtain the results in the figures above we choose to use a network made of 6 cameras, initialized with bad initial conditions. In fact, also here cameras 2 through 6 are initialized with the identity value (camera 1 is initialized with its true pose). Moreover, the setup is noise-free.

On the left side of the previous page, we can see that the use of quaternions on the cost function is really effective. The fact that we have no noise on the relative measures ensures that the cost function converges toward zero.
On the right side, we have the results obtained with the use of the TV algorithm. It is clear that the initial and final rotation estimates are completely different and the cost function value is very far from zero.

A comparison with the DDQL algorithm is difficult since we are considering only the rotation estimation.

Chapter 5

Conclusions

We presented a distributed algorithm able to optimally solve the localization problem for camera networks. The original part of this work is the use of unit quaternion to represent the pose of the camera on the network and the application of hybrid ideas for the further optimization of the initial algorithm.

Even if only the rotation estimation is considered, we can say that Alg. 1 works very well and is robust with respect to the white Gaussian noise standard deviation σ applied to initial estimate and relative quaternion measures.
Indeed, we were able to prove the effectiveness of the use of the quaternion paradigm for the estimation of the orientation of cameras in sensor networks, with different dimensions, when both the initial conditions and the relative measures are perturbed.

Regarding the hybrid approaches, we were not able to apply them. The first one, described in Alg.2, does not work due to the symmetry of the cost function J. The second one, described in Alg.3, should work but despite the numerous simulations done, we were not able to find a set of initial conditions that would allow its use (*i.e.* a set of conditions that allows a slower convergence of the cost function).

To conclude, in every realistic scenario, Alg. 1 is able to guarantee rapid convergence towards the optimal solution, without getting stuck in a local minimum.

A future development of this work is to consider the translation estimation. In this way, the algorithm would be able to estimate the full pose of each camera. The innovation of applying a hybrid law could be included in this part of the work.

Bibliography

[1] R. Goebel, R. G. Sanfelice, and A. R. Teel, "Hybrid dynamical systems," *IEEE control systems magazine*, vol. 29, no. 2, pp. 28–93, 2009.

[2] I. Fedorov, N. Lawal, B. Thőrnberg, H. Alqaysi, and M. O'Nils, "Towards calibration of outdoor multi-camera visual monitoring system," in *Proceedings of the 12th International Conference on Distributed Smart Cameras*, ICDSC '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[3] B. D. Olsen and A. Hoover, "Calibrating a camera network using a domino grid," *Pattern Recognition*, vol. 34, no. 5, pp. 1105–1117, 2001.

[4] M. Khan, E. Dannoun, M. Nofal, and M. Mursaleen, "Significance of camera pixel error in the calibration process of a robotic vision system," *Applied Sciences*, vol. 12, p. 6406, 06 2022.

[5] G. Michieletto, A. Cenedese, L. Zaccarian, and A. Franchi, "Hierarchical nonlinear control for multi-rotor asymptotic stabilization based on zero-moment direction," *Automatica*, vol. 117, p. 108991, 2020.

[6] L. Varotto, M. Fabris, G. Michieletto, and A. Cenedese, "Distributed dual quaternion based localization of visual sensor networks," in *2019 18th European Control Conference (ECC)*, pp. 1836–1841, 2019.

[7] R. Tron and R. Vidal, "Distributed image-based 3-d localization of camera sensor networks," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 901–908, IEEE, 2009.

[8] J. Yan-Bin, "Quaternions and rotations," *Com S*, vol. 477, p. 577, 2014.

[9] B. Bollobás, *Modern graph theory*, vol. 184. Springer Science & Business Media, 1998.

[10] L. Song, W. Wu, J. Guo, and X. Li, "Survey on camera calibration technique," in *2013 5th International conference on intelligent human-machine systems and cybernetics*, vol. 2, pp. 389–392, IEEE, 2013.

[11] W. Qi, F. Li, and L. Zhenzhong, "Review on camera calibration," in *2010 Chinese Control and Decision Conference*, pp. 3354–3358, 2010.

[12] R. Alur, T. Dang, and F. Ivančić, "Predicate abstraction for reachability analysis of hybrid systems," *ACM Trans. Embed. Comput. Syst.*, vol. 5, p. 152â199, feb 2006.

[13] R. Tron, *Distributed Optimization on Manifolds for Consensus Algorithms and Camera Network Localization*. PhD thesis, USA, 2012. AAI3536513.

[14] R. Tron, R. Vidal, and A. Terzis, "Distributed pose averaging in camera networks via consensus on se(3)," in *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*, pp. 1–10, 2008.

List of Algorithms

List of Figures