Universitat Politècnica de Catalunya

Master's degree in

Automatic Control and Robotics

Master's Thesis

Hierarchical Task Control through Lexicographic SemiDefinite Programming

Application to Unmanned Aerial Manipulators

Josep Martí Saumell

Director: Dr. Angel Santamaria-Navarro
Rapporteur: Prof. Carlos Ocampo-Martinez

Academic Course 2017/18

September 2018

Abstract

In this thesis, we present a new method to simultaneously fulfill several tasks with a robot. We take advantage of redundant robots, in terms of degrees-of-freedom, and we are able to prioritize between tasks by expressing the control law as a lexicographic semidefinite programming, which consists on a hierarchized group of optimal problems. These optimal problems are formulated through linear matrix inequalities, enabling the definition of equality tasks (commonly done in the literature) and also inequality tasks. With this formulation, we describe the problem in a compact form that eases its extension. Moreover, this structure using linear matrix inequalities allows us to use common linear algebra tools, hence to use fast specialized solvers, making feasible a real-time solution of the problem for highly redundant robots. We express the tasks in the acceleration domain, obtaining smooth joint references. The proposed approach is applied to autonomously drive aerial manipulators, providing the analysis of simulations and real case studies in the second part of this work. For the sake of completeness, we also include in the appendices complete descriptions of general formulation procedures as well as a complete and didactic practical example. The code produced within this work is made publicly available for the benefit of the community.

Acknowledgements

A la meva companya Montse, les meves germanes Núria i Aleth i també a la meva mare Lurdes i al meu pare Pere.

Als meus tutors Àngel i Carlos.

També a l'Institut de Robòtica Industrial on he realitzat aquesta tesi.

This work has been partially supported by the EU H2020 project AEROARMS (H2020-ICT-2014-1-644271) and by the Spanish State Research Agency through the María de Maeztu Seal of Excellence to IRI (MDM-2016-0656).

Contents

Acronyms

CLIK	Closed-loop inverse kinematics
DoF	Degrees of freedom
IK	Inverse kinematics
IMU	Inertial measurement unit
IRI	Institut de Robòtica i Informàtica Industrial de Barcelona (CSIC-UPC)
LMI	Linear matrix inequalities
LQP	Lexicographic quadratic programming
LP	Linear programming
LS	Least squares
QP	Quadratic programming
ROS	Robotic Operating System
SDP	Semi-definite programming
UAM	Unmanned aerial manipulator
UAV	Unmanned aerial vehicle

List of Figures

1. Introduction

In early designs, robots were created for specific purposes that required the execution of simplistic movements in well-controlled environments. Nowadays, technology is mature enough to allow more complex robotic designs. As a consequence, the complexity of the actions expected for the robots has also increased: we now ask robots to accomplish several tasks simultaneously and often some of them can even be contradictory. Thus, it has emerged the need for frameworks to describe qualitatively how the robot's behavior should be and to let the robotic system decide by itself, *i.e.,* multiple task control. Precisely, this work aims at offering one of these frameworks.

If we understand a robot as a chain of rigid bodies (*links*) connected by means of articulations (*joints*), we refer to the number of *degrees of freedom* (DoF) as the amount of independent variables required to define the situation of every joint. The number of free DoFs at a particular configuration defines the ability of a robot to perform certain tasks, called maneuverability. Hence robots with higher number of DoFs will be able to accomplish more complex tasks. For example, the task of positioning and orientating a body in the space requires 6 DoFs (3 for position and 3 for orientation). In those cases where the number of robot DoFs is higher than the required by the task, we can talk about a redundant robot. Thanks to their high maneuverability, redundant robots are able to perform multiple tasks at the same time. This thesis is about controlling a highly redundant robot in order to perform some pre-defined tasks simultaneously.

In robotics, solving the *inverse kinematics* (IK) problem means to find the joint values of the kinematic chain that reaches a specific robot end effector pose. When talking about robots and tasks, solving the IK problem entails finding the joint values over time to accomplish a specific task. If we operate with redundant robots, there is not a single IK solution but a set of possible joint value combinations, hence we can reach the same robot configurations with different joint values. This DoF overdetermination can be exploited in order to accomplish other tasks at the same time. However, there exist situations in which even with a redundant robot it is not possible to solve all required tasks at the same time, and one should find ways to enforce that at least the critical ones are fulfilled. Thus, it is interesting to find solutions to apply task hierarchies in order to prioritize the accomplishment of these tasks depending on chosen criterias.

Although there exist in the literature different approximations to solve the IK problem for several hierarchized tasks using redundant robots (reviewed in Section 2.1), in this thesis the IK problem is tackled by solving a Lexicographic *semi-definite programming* (SDP)

problem, where the term lexicographic stands for solving a sequence of single-objective optimization problems. We also provide simple examples, to help the reader understanding. Moreover, we provide a series of appendices with related material, including a simple but complete example using the main mathematical formulations of the thesis. Moreover, for the benefit of the community, we made the code available.

1.1 Motivation

For those not familiarized with this kind of robots, UAMs are composed by 2 robotic parts: one is a flying platform -generally a multicopter- while the other is a robotic arm manipulator attached to the platform. Since we are considering two robots as a whole part, there is a considerable amount of DoFs to be controlled. Having a framework that eases the control of its DoFs by specifying tasks in a natural sense, allowing to add or remove tasks modularly is crucial. The computation of the joint references enabling this control framework is commonly known as trajectory generation.

Trajectory generation for highly redundant robots is an area that has been investigated since the late 80's of the past century. It tries to find methods for harnessing all the DoFs of a robot while accomplishing one or several tasks. This trajectory generation is desirebale to be performed on-line, *i.e.,* at every time step the implemented algorithm must provide the low-level joint controller with new reference values that ensure the accomplishment of the tasks. Notice how on-line computation requires light algorithms and a considerable computational burden.

A common approach to trajectory generation is by means of mathematical optimization. There are several algorithms to analytically solve the optimization problem developed for generating the trajectory. However, these methods hinder the fulfillment of some specific tasks and, when introduced, these tasks make the formulation substantially more complicated. In recent years, thanks to the improvement on computational capacity of computers, together with the appearance of fast and efficient numerical solvers, the application of numerical optimization on-line has become a hot topic.

This master's thesis arises within the mobile robotics research group and, more specifically, from the work conducted during the last years by Dr. Angel Santamaria-Navarro. During this period, UAMs have been introduced to *Institut de Robòtica i Informàtica Industrial de Barcelona* (IRI) mobile robots laboratory, encouraging the appearance of several research works, such as analitical hierarchical control laws for equality tasks, or a trajectory generation algorithm through quadratic programming, where the tasks priorities are combined as a weighted sum of cost functions. The challenge and motivation for

conducting this master's thesis is to provide a trajectory generation method able to handle several hierarchical tasks, considering inequality constraints, and specified in compact way.

In this thesis, we propose a method to hierarchically solve the IK problem for different tasks by means of lexicographic semi-definite programming problems, which imply the use of *linear matrix inequalities* (LMI).

The main advantages of this formulation with respect to those already reported in the literature for UAMs are the following:

- We are able to describe the problem in a compact form.
- We can solve tasks hierarchically by means of optimization procedures, which allow us to introduce inequality tasks in the hierarchical structure of the problem.
- The LMI expression allows for a fast optimization using specialized solvers. Through the LMI structure, these solvers can use common linear algebra tools to efficiently solve SDP problems, which makes feasible a real-time solution for robots with a high number of DoFs.
- The lexicographic structure, *i.e.,* a nested group of single-objective optimizations, allow us to create a task priority manager to manage and exchange priorities of tasks online, without the dependence of weights or the need for control reference smoothers (*e.g.,* in the case of analytical solutions using the projection of subtasks into the null space of the tasks higher in the hierarchy, the control reference computed has to be smoothed when the tasks change their priorities [1] or [2]). This is of high interest considering the reduction of parameters (weights) to be tunned for each particular application.

1.2 Objectives

The main objectives of this master's thesis are:

- Review the solutions for trajectory generation available in the literature.
- Develop an algorithm to impose strict hierarchy between tasks.
- Consider either equality and/or inequality tasks expressed in the acceleration domain.

- Formulate the problem as an LMI-based optimization problem, *i.e.,* an SDP problem in order to use the algorithm on-line.
- Implement the algorithm using LMI dedicated numerical solvers.
- Validate the approach through partial (*i.e.,* singular tasks with simplified robot models) and complete simulation case studies.
- Perform real robot experiments using *Robot Operating System* (ROS) compatible code within a real UAM, performing several tasks in a single mission.

1.3 Notation

Within this thesis, we take advantage of two main notations, for the following reasons.

When describing mathematical tools, *e.g., least squares* (LS) approach, we use a generic notation. This notation is helpful to describe generic problem formulations with commonly used variable names. Thus, we expect the reader to quickly identify the involved mathematical agents as the namings are quite standard in the literature. As an example, here we have generic variable names such as $\boldsymbol{x}$, $\boldsymbol{A}$ or $\boldsymbol{b}$.

Another notation is used for the generic variable names to the target field of this thesis: robotics. We believe that using identifiable variable names in the resulting formula developments would help the reader, not only to better picture the formulation impact but also to better implement the methods here presented. In this case we use variable names such as $\boldsymbol{q}$, $\boldsymbol{J}$ or $\boldsymbol{\sigma}$ to describe robot joint positions, Jacobians or task variables, respectively.

In those chapters of the thesis where important mathematical tools are required, we start defining them with the generic notation. Then, we specialize that notation and move to the robotics definitions depending on the particularities of the developments considered. In this way, even though the robotics developments can be complex, the user is always able to indetify the elements of the original mathematical tools and keep track of its rules. Moreover, we can take advantage of variable substitutions to be concise in the developments.

In Section 2.5.3, we describe the procedure to convert our formulation, expressed in the continuous-time domain, into a discrete-time domain one. During the whole document, dots on top of letters are used to indicate continuous time-derivatives. For example, if $\boldsymbol{q}$ is used to express joint positions, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ would indicate joint velocities and joint accelerations, respectively. For the sake of the fluidity in the reading of the work, to

express velocity or acceleration in the discrete-time domain, we have added a subscript k in the continuous-time domain variables. Therefore, a expression in discrete-time domain like $\ddot{\boldsymbol{q}}_k$ should be interpreted as: *joint accelerations evaluated at instant k*.

During the thesis the symbols $\prec$, $\preceq$, $\succ$ and $\succeq$ are widely used. When used with vectors they stand for component-wise inequalities, *e.g.,* , $\preceq$ stands for component-wise *smaller or equal than* ($\leqslant$). However, when used with matrices they stand for negative definite, negative semidefinite, positive definite and positive semidefinite respectively.

1.4 Outline

This thesis is structured with the following chapters:

Chapter 2: Background This chapter starts presenting the state-of-the-art related with the main topics developed in this work. Besides, it also describes how this thesis is positioned with respect to this state-of-the-art. This is followed by theoretical concepts that will be used as a base in the following Chapter 3. Within these concepts, we explain the redundancy in robotics, we continue with a common and general way of formulating tasks by means of their task Jacobians. We also describe the different approaches to solve the IK problem, either using an analytical solution or by means of mathematical optimization. At the end of the chapter the *closed loop inverse kinematics* (CLIK) is defined with some considerations about its implementation using acceleration. As we implement the presented algorithm in digital robot controllers, the last section of this chapter is devoted to explain how to express the equations within the discrete domain.

Chapter 3: Hierarchical task control through Lexicographic SDP This chapter is the core of this work as it presents the value proposition of the thesis. We start by detailing how to transform the state-of-the-art *quadratic programming* (QP) problem equations so they can be solved by means of an SDP problem. The last section encompasses the task resolution using SDP with the algorithm used in further experiments.

Chapter 4: Application to UAMs This chapter encompasses the application to UAMs of the theory explained in Chapter 3. We begin with a detailed description and formulation of the tasks implemented in the UAM. Afterwards, we offer a technical description of the robot used: the Kinton UAM. In the last part of the chapter, we present the experiment set, the priority manager and the validation of the proposed approach

through real robot experiments. For the sake of conciseness, the results of the complete UAM simulations are not included in this document beacuse the real experiments are better to analize and extract the conclusions of this work.

Concluding remarks In this final chapter, conclusions and final remarks are drawn. We also point out the possible future work within the field of UAMs and trajectory generation.

Appendices Although being in the appendices, it is worth to mention its content. In the first appendix one can find an explanation about some mathematical tools, which have been used for developing the trajectory generation in an SDP framework. Moreover, to better introduce the reader to this master's thesis, Appendix B presents a guided application to a 4-link planar manipulator of the algorithm developed here. The reference to the code used in this chapter is also given for the sake of completeness.

2. Background

The content of this chapter is related with the work done in the fields of IK and trajectory generation existing in the literature. It is divided in two main parts: first, we detail, qualitatively, the state-of-the-art related with the main topics developed in this work; second, we describe the common analytical tools used in this field. In the latter, we introduce the concept of redundancy in the field of robotics, followed by the *task* notion with a mention on how to obtain the task Jacobian. Considering the purpose of this thesis, in the last part of this chapter we define several task control laws to solve the IK problem preserving a certain hierarchy between tasks, with a sequential explanation of the different methods explored. We start from the traditional LS analytical solution and we end up presenting two different approaches that use optimization problems with a QP structure.

2.1 State of the art

In the field of robot manipulators, finding effective methods to solve the IK problem has always been a fundamental topic of research. In a basic manner, that is finding which values should the joints have to position the manipulator end-effector in a desired pose. Theoretically, only 6 DoFs are required to achieve any end-effector position and orientation: a solution in the 3-dimensional *Special Euclidean* group, $SE(3)$. When talking about common robot manipulators, nevertheless, this assertion only holds for a certain regions of the robot's workspace. Due to its constructive geometry, there are some configurations (*i.e.,* a specific combination of values for every robot's joint) where the robot's functionality is reduced. These problematic configurations mostly occur when the robot has aligned two or more rotation axis. This phenomena is commonly known as singularity. This is an undesired situation in the vast majority of applications, and a possible solution is found by adding extra links and joints to the robot. This strategy is explained in [3].

As in the case of positioning the robot end-effector, a task can be understood as a portion of the robot's function, aimed to accomplish a specific dynamic or kinematic goal. In [4] and [5], the authors introduce this concept of task and discuss the use of extra DoFs to perform not only one task but many. For example, the robot's main function could be to move the end-effector from one point to an other, aiming at some object manipulation, and in order to accomplished this target it would require the achievement of several tasks simultaneously. We might want the robot end-effector to accomplish with a pose task.

That is to follow a specific trajectory (positioning task) it with a specific orientation (orientating task). Besides, we could also specify other tasks such as low energy consumption, an obstacle avoidance task, etc. Since then, these task concepts have been adopted by the research community working on the IK problem.

In [4] the IK problem is solved as a LS optimization problem. It uses its analytical solution using the Jacobian's pseudo-inverse to project an error computed in the task space to the joint space. It also discusses the procedure to prioritize among tasks using the same technique, *i.e.,* in case of running out of DoFs the method ensures that the tasks with higher priority will be firstly accomplished over those with lower priority. This priorization method is based on projecting the resulting joint velocities of a secondary task onto the null space of the Jacobians of those tasks with higher priority. This framework is now a reference in the robotics field and it is explained in several generalist books like [6] or [7].

In order to fulfill a task, we must ensure that the variables describing this task take specific values. Depending on the nature of the task, these values could be either unique or form a set of values. The first case would be an *equality task*, *e.g.,* a positioning task, while the latter case would be an *inequality task* like keeping safety distances to obstacles (obstacle avoidance) or to the joint limits. The treatment of inequality tasks is of special importance due to the fact that there is no method to find an analytical solution to this kind of tasks. For that reason, alternative methods were found to introduce them as equality tasks. In [8] and [4] is proposed to convert the inequality task into a potential field that is minimized as a secondary task. Minimizing these potential fields would move solutions away from obstacles (to avoid obstacles [4]) or from conflicting configurations (to avoid singularities [5]), whilst ensuring the accomplishment of the primary task, *i.e.,* pose tracking.

All the methods described so far have the nature of mathematical optimization programs. The improvement in computational power has stimulated the appearance of methods that, instead of trying to solve these LS problems analytically, make use of numerical solvers specially created to solve QP problems. This allows a more natural expression of the inequality tasks as they can be added as inequality constraints to the optimal problem. This is the case of [9] and [10] where, besides of introducing the inequality tasks in a more natural form, a hierarchy among them is also given. They consider a hierarchical QP problem also known as a *lexicographic quadratic programming* (LQP) problem. In this case, to achieve the following point in the trajectory of every joint, several nested QP problems must be solved (one for every level of the hierarchy). To preserve the hierarchy among tasks, solutions of the QP problems of tasks higher in the hierarchy are added as constraints to the lower priority QP problems. This way of proceeding and interacting with later optimizations is known as lexicographic optimization ([11], [12]). All these approaches add slack variables to every inequality task in order to avoid unfeasibility issues and to relax the constraints when they cannot be fulfilled.

There are several facts that can increase considerably the computational burden, *e.g.,* the number of tasks to be solved (linked with the number of hierarchy levels) or the number of robot DoFs. Reducing this computational burden has always been a field of interest. In [13], the authors use a pseudo-inverse analytical solution to solve one single optimization problem with all the tasks already hierarchized inside the objective function. However, they require the calculation of the task Jacobian pseudo-inverse. They also explain a method to deal with hierarchized inequality tasks based on the *active set strategy* [14].

The active set strategy is an iterative procedure that begins by choosing some inequality tasks and converting them into equalities, assuming that they are all active (an inequality is said to be active when it is fulfilled by an equality). Then, the optimization problem with only equality constraints is solved (nowadays, there exist dedicated numerical solvers that uses this strategy [14]). The iteration ends checking all inequality tasks as follows: if the task was included in the set of converted tasks (inequalities to equalities) but the minimum can be reached without violating it, then this task is removed; on the other hand, if the task was not included but is being violated by the achieved solution, this task is added. If none of these situations happen, the algorithm comes to the end. This method is further explained in [15].

Another close approach is [16], where the number of optimization problem solutions is reduced to one. In this work, a weighted sum of all equality tasks is set as cost function of a QP problem. All inequality tasks are set as problem constraints. A drawback of this approach resides in the hierarchy of tasks, which is only given to equality tasks by means of assigned weights, requiring also a fine tuning work. Moreover, no hierarchy neither strict nor by means of weights is given to inequality tasks. A different point of [16] is the method to express any task in the acceleration domain. Notice that all previously reviewed works describe the tasks in the velocity domain. However, there are several advantages of considering the accelerations when solving the IK problem. There is an explicit implication of the acceleration of the joints in the dynamic equations of the robot. Therefore, if one wants to consider the dynamics, it makes sense to express tasks in the acceleration domain (*e.g.,* [4]). Even though in our case we perform kinematic control, by doing it using the accelerations we will also contribute in a better performance of the robot dynamics with smooth trajectories in the position domain [17].

The hierarchical task-based framework for solving the IK problem makes sense when the robot is redundant. The vast majority of papers cited in this section apply their algorithms to either human-like robot arms or fully humanoid robots. However, during the last years it appeared a new field of robotics called aerial manipulation where *Unmanned Aerial Vehicles* (UAVs) equipped with a robotic arm are used to manipulate objects, namely *unmanned aerial manipulators* (UAMs). This is the case of [18], where a hierarchical task control is performed considering equality tasks, and solving the IK problem using the analytical solution with the Jacobian projection method. Similarly, [1] uses the same

approach, but this time highlighting a secondary task to drive the end-effector by means of an uncalibrated image-based visual servo strategy.

The pure IK problem assumes the robot and its operation to be ideal. That is to assume that the geometry is perfect (no misalignment or deviations in its dimensions) and that no perturbation affects the robot (*i.e.,* the robot is considered able to accomplish with the computed joint references within the time steps used for those computations). These assumptions are not actually true and, therefore, it makes sense to compute a task error comparing the desired value of a specific task with its real value. This is known as the *closed-loop inverse kinematics* (CLIK) problem. The framework stated above for the IK problem (and the reviewed works) can be easily extended to consider a CLIK scheme, slightly modifying the tasks expressions. This was already done in [4]. Afterwards, several studies regarding the stability of these closed loop tasks have been carried out, *e.g.,* in [19] or [20] by means of Lyapunov theory, or estimating the region of attraction in [21]. The controllers with a CLIK scheme also deal with numerical and sampling issues when implementing discrete versions of these algorithms in micro-controllers ([21], [22]).

Given the overviewed state-of-the-art, we present in this thesis an approach to solve the CLIK problem, considering not only equality tasks but also inequalities. As UAMs have become a reality, it seems interesting to bring the state-of-the-art of trajectory generation for highly redundant robots to the field of aerial manipulation and, to demonstrate the viability of the presented approach for real applications, a particular case study is shown to control a real UAM robot. We do not compute the analytical solution for the hierarchical control (which projects the solution of a task into the null space of an other task with a higher hierarchy) beacuse it considers only equality tasks and includes a lot of parameters (weights) to be tunned for each application. For similar reasons, we are not interested in solving the problem with a QP approach which can be difficult to tune (*e.g.,* [16]). Instead, the method presented in this thesis is defined with LMIs, bringing compact expressions that do not require much parameters to set the tasks and hierarchies. Moreover, we solve the problem as an SDP, allowing the use of very fast specialized and dedicated solvers.

2.2 Redundancy concepts

A robot is kinematically redundant when it is composed of a number of DoFs greater than the number of variables necessary to perform a specific task. Hence, let us define the following space dimensions:

- m: Dimension of the operational space (*e.g.,* a robot moving in 2D space has $m = 3$, 2 for the plane cartesian coordinates and 1 for the orientation).

- j: Dimension of the joint space (*i.e.,* DoFs of the robot).
- r: Dimension of the task to be performed by the robot (*e.g.,* positioning a robot without orienting it in the 2D space has $r = 2$).

As a complete example, consider a robot with 4 joints (rotational joints), moving in 2D space and trying to perform a positioning task, then $m = 3$, $j = 4$ and $r = 2$. Note how in this example we have a redundant robot with 2 extra DoFs that could be used to perform another task.

Talking about redundancies, according to the previous definitions we consider the following typologies:

- **Kinematic redundancy:** $j > m$ (*e.g.,* a human arm is kinematically redundant since it has 7 DOFs).
- **Functional redundancy:** $j > r$. This situation exists when the robot and the operational space have the same dimensions ($m = j$), thus the robot is not kinematically redundant. However, the task to be performed does not require that amount of DoFs ($j > r$), hence the robot has a functional redundancy. For example, let us consider a planar manipulator with three (rotational) joints. If we want to position and orientate the end-effector in the plane, there will be no redundancy of any kind ($m = j = r$). However, if we only want to position that end-effector, we will have a *functional redundancy* ($m = j > r$).

These redundancy concepts are here described for the sake of clarity and completeness, and to familiarize the reader with space, task and robot dimension concepts. However their differentiations are not highlighted in the rest of this work.

2.3 Generic task formulation

Our goal is to solve several tasks taking advantage of the robot redundancy. To do so, for each single task we will compute the result of a cost function, namely $\boldsymbol{\sigma}$, and then obtain the robot joint values $\boldsymbol{q}$. This can be generically expressed with

$$\boldsymbol{\sigma} = \boldsymbol{f}(\boldsymbol{q}) \,, \tag{1}$$

where $\boldsymbol{\sigma} \in \mathbb{R}^r$ are the values of the task variables (*e.g.,* if we want to position and orient the arm end-effector, $\boldsymbol{\sigma}$ is the resulting pose of the end-effector). $\boldsymbol{q} \in \mathbb{R}^j$ are the joint

values defining the robot configuration, which makes the robot accomplish with the task values $\boldsymbol{\sigma}$ (*i.e.,* $\boldsymbol{q}$ are the joint values).

Differentiating (1) with respect to time, we obtain first-order differential kinematics (*e.g.,* in the case of just positioning the end-effector, it corresponds to the end-effector velocities, $\boldsymbol{v}_e = \dot{\boldsymbol{\sigma}}$), which can be expressed as

$$\dot{\boldsymbol{\sigma}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q}) \dot{\boldsymbol{q}} \,, \tag{2}$$

where we find the so called task Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$. Notice how in (2) the previous dimensions hold, therefore $\dot{\boldsymbol{\sigma}} \in \mathbb{R}^r$, $\dot{\boldsymbol{q}} \in \mathbb{R}^j$ and $\boldsymbol{J} \in \mathbb{R}^{r \times j}$.

In summary, a robot joint's trajectory can be generated subject to the accomplishment of several tasks, such as following a specific predefined path or avoiding collisions, and each of these tasks will have a task Jacobian (2). Although these tasks can consider both robot kinematics and dynamics, in this thesis we focus on the formulation to effectively combine them and the ones presented here are based on robot kinematics.

2.3.1 Task Jacobian

Here we explain the process of obtaining the Jacobian matrix of a specific task. For the sake of simplicity, we use a simple and easy to follow task example: positioning the end-effector of a 2-link planar manipulator. This robot is the same as the one defined in [23], and its simple design is shown in Figure 1.

The 2-link manipulator has a coordinate frame $\{2\}$ attached to its end-effector, which moves along with the second link. We consider a base fixed frame $\{0\}$ with its origin on the center of the fixed articulation, its first horizontal coordinate $\bar{X}$ pointing to the right of the picture and the second vertical coordinate $\bar{Y}$ pointing upwards. Then, the homogeneous transformation matrix from the base frame $\{0\}$ to the frame attached at the end-effector $\{2\}$ is

$${}^0_2\boldsymbol{T} = \begin{bmatrix} c_{12} & -s_{12} & l_2 c_{12} + l_1 c_1 \\ s_{12} & c_{12} & l_2 s_{12} + l_1 s_1 \\ 0 & 0 & 1 \end{bmatrix} , \tag{3}$$

being

$$\begin{aligned} c_{12} &= cos(q_1 + q_2) = cos(q_1)cos(q_2) - sin(q_1)sin(q_2) \,, \\ s_{12} &= sin(q_1 + q_2) = sin(q_1)cos(q_2) + cos(q_1)sin(q_2) \,. \end{aligned} \tag{4}$$

The position of the end-effector ($\boldsymbol{\sigma}_e$) is the origin of the frame $\{3\}$, hence, ${}^3\boldsymbol{\sigma}_e = [0, 0, 0]^\top$.

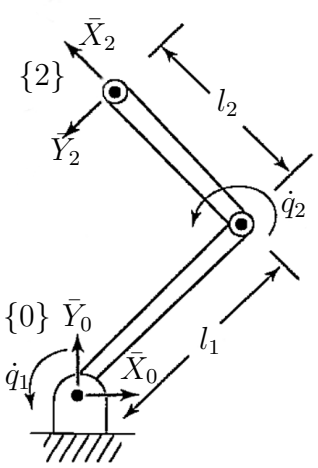


Figure 1: Two-link planar manipulator used in the example to show the computation of a task Jacobian.

Notice the nomenclature used in this part to specify the coordinate frame where a variable is expressed, namely X, as upper left superscript, *e.g.*, ${}^{X}\boldsymbol{a}$.

If we want the origin in terms of the fixed frame $\{0\}$, we have to transform the frame using the homogenous transformation matrix (3), such as

$$\begin{bmatrix} {}^{0}\boldsymbol{\sigma}_e \\ 1 \end{bmatrix}^{\top} = {}^{2}_{0}\boldsymbol{T} \begin{bmatrix} {}^{2}\boldsymbol{\sigma}_e \\ 1 \end{bmatrix}^{\top} = \begin{bmatrix} l_2 c_{12} + l_1 c_1 \\ l_2 s_{12} + l_1 s_1 \\ 1 \end{bmatrix}. \tag{5}$$

The robot Jacobian matrix relates the joint velocities (in this case $\dot{q}_1$ and $\dot{q}_2$) with the end-effector velocities ($\dot{\boldsymbol{\sigma}}_e$) expressed in $\{2\}$. Therefore, it can be written as

$$\dot{\boldsymbol{\sigma}}_e = \begin{bmatrix} \boldsymbol{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{J}_v \\ \boldsymbol{J}_\omega \end{bmatrix} \dot{\boldsymbol{q}}\,, \tag{6}$$

where $\boldsymbol{v}_e$ and $\boldsymbol{\omega}_e$ are the linear and angular velocities of the end-effector, respectively. Besides, $\boldsymbol{J}_v$ and $\boldsymbol{J}_\omega$ are the elements (rows) of the robot Jacobian that relate joint velocities with linear and angular velocities of the end-effector. Joint states are expressed as $\boldsymbol{q} = [q_1\, q_2]^{\top}$. Hereafter, we omit the superscript indicating the coordinate frame for the sake of clarity and conciseness.

If we differentiate the position with respect to time, using the chain rule, we get the

following expression:

$$\boldsymbol{v}_e = \frac{\partial \boldsymbol{\sigma}_e}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial t} = \begin{bmatrix} \frac{\partial \boldsymbol{\sigma}_e^1}{\partial q_1} & \frac{\partial \boldsymbol{\sigma}_e^1}{\partial q_2} \\ \frac{\partial \boldsymbol{\sigma}_e^2}{\partial q_1} & \frac{\partial \boldsymbol{\sigma}_e^2}{\partial q_2} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}, \tag{7}$$

where the 2×2 matrix formed by partial derivatives of the position with respect to the joint variables is $\boldsymbol{J}_v$, which is computed differentiating (5), we get

$$\boldsymbol{J}_v = \begin{bmatrix} -l_2 s_{12} - l_1 s_1 & -l_2 s_{12} \\ l_2 c_{12} + l_1 c_1 & l_2 c_{12} \end{bmatrix}. \tag{8}$$

And the part of the Jacobian related to the rotation variable is the scalar

$$\omega_e = \dot{\theta}_1 + \dot{\theta}_2, \text{ and hence } \boldsymbol{J}_\omega = \begin{bmatrix} 1 & 1 \end{bmatrix}. \tag{9}$$

Now, we can express the full Jacobian as a 3×2 matrix with (6) as the following:

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} -l_2 s_{12} - l_1 s_1 & -l_2 s_{12} \\ l_2 c_{12} + l_1 c_1 & l_2 c_{12} \\ 1 & 1 \end{bmatrix}. \tag{10}$$

2.4 Task resolution

Generally speaking, as the task cost $\boldsymbol{\sigma}(t)$ is usually provided, the inverse kinematic problem consists in determining a joint trajectory $\boldsymbol{q}(t)$ which satisfies (1) at each time step t. This problem can be solved at the differential level using (2). Therefore, the solution will be given in terms of joint velocities. Finding an exact solution for this problem is not always possible. This leads us to think in an optimization problem that finds the closest solution. Minimizing the squared 2-norm, we can state the following LS problem:

$$\min_{\dot{\boldsymbol{q}}} . \, ||\boldsymbol{J}(\boldsymbol{q})\, \dot{\boldsymbol{q}} - \dot{\boldsymbol{\sigma}}||_2^2 \,. \tag{11}$$

2.4.1 Traditional least squares approach

In this section, we describe how to find the analytical solution to LS problem considering also the possibility to prioritize among tasks (task hierarchy). For the sake of clarity and consistency, we will use in this section the standard mathematical notation for a least squares problem, denoted as

$$\min_{\boldsymbol{x}} \, ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2 \,, \tag{12}$$

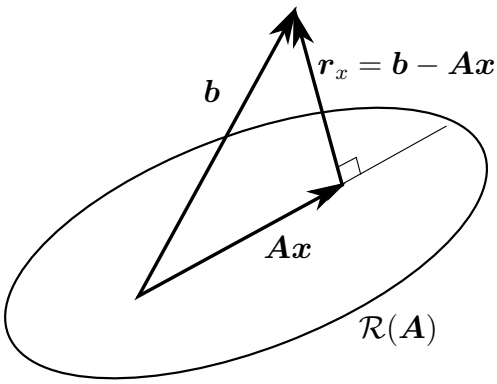


Figure 2: Graphical representation of the least-squares problem, with the manifold $\mathcal{R}(\boldsymbol{A})$ defined by the range of $\boldsymbol{A}$. The vector $\boldsymbol{r}_x$ is the residual between $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{b}$, which must be orthogonal to $\mathcal{R}(\boldsymbol{A})$ ($\boldsymbol{r}_x \perp \mathcal{R}(\boldsymbol{A})$).

where the least squares solution can be understood as the values of $\boldsymbol{x}$ that multiplied by the matrix $\boldsymbol{A}$ give the closest values to $\boldsymbol{b}$, *i.e.,* the residual $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ is close to $\mathbf{0}$. This residual is squared to ensure convergence when minimizing it (*i.e.,* concave function with a global minima) and continuity in its derivative.

To better analyze the problem, let $\mathcal{S}$ be the set of solutions of the LS problem, defined as

$$\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^j \mid \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \rightarrow min\}\,. \tag{13}$$

Defining the residual $\boldsymbol{r}_x = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, we can say that

$$\boldsymbol{x} \in \mathcal{S} \Leftrightarrow \boldsymbol{A}^\top \boldsymbol{r}_x = \mathbf{0} \Leftrightarrow \boldsymbol{r}_\mathbf{x} \perp \mathcal{R}(\boldsymbol{A})\,, \tag{14}$$

where $\perp$ implies orthogonality and $\mathcal{R}(\boldsymbol{A})$ is the manifold defined by the range of $\boldsymbol{A}$. Recall that the range of a matrix (also called *column space*) is the set of all linear combinations formed by its column vectors. Therefore, when we do $\boldsymbol{A}\boldsymbol{x}$ we are actually doing a linear combination of the columns of $\boldsymbol{A}$ and thus, $\boldsymbol{A}\boldsymbol{x} \in \mathcal{R}(\boldsymbol{A})$.

We can see a scheme of the last implication of (14) in Figure 2. When $\boldsymbol{b}$ is not exactly on $\mathcal{R}(\boldsymbol{A})$, there will exist a residual vector $\boldsymbol{r}_x$ which will be minimum when it is orthogonal to the manifold $\mathcal{R}(\boldsymbol{A})$.

To understand the implication $\boldsymbol{A}^\top \boldsymbol{r}_x = \mathbf{0}$ in (14), we first consider each column vector of $\boldsymbol{A} = \begin{bmatrix}\boldsymbol{a}_1 & \cdots & \boldsymbol{a}_n\end{bmatrix}$. This allow us to write the expression as the dot product ($\cdot$) between $\boldsymbol{r}_x$ and every column vector of $\boldsymbol{A}$

$$\boldsymbol{A}^\top \boldsymbol{r}_x = \begin{bmatrix}\boldsymbol{a}_1 \cdot \boldsymbol{r}_x, & \cdots & \boldsymbol{a}_n \cdot \boldsymbol{r}_x\end{bmatrix}^\top . \tag{15}$$

If $\boldsymbol{r}_x$ is orthogonal to $\mathcal{R}(\boldsymbol{A})$ implies that it is also orthogonal to every vector composing $\mathcal{R}(\boldsymbol{A})$. Therefore, orthogonal to $\boldsymbol{A}$, thus

$$\boldsymbol{A}^\top \boldsymbol{r}_x = \begin{bmatrix} \boldsymbol{a}_1 \cdot \boldsymbol{r}_x, & \cdots & \boldsymbol{a}_n \cdot \boldsymbol{r}_x \end{bmatrix}^\top = \boldsymbol{0}\,. \tag{16}$$

Following with (16) and expanding it, it leads us to the analytic solution for the least squares problem

$$\boldsymbol{A}^\top \boldsymbol{r}_x = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0} \rightarrow \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{b} \rightarrow \boldsymbol{x} = \boldsymbol{A}^+ \mathbf{b}\,, \tag{17}$$

being $\boldsymbol{A}^+ = \left(\boldsymbol{A}^\top \boldsymbol{A}\right)^{-1} \boldsymbol{A}^\top$ the left *More-Penrose* pseudoinverse. As $\boldsymbol{x}$ is minimum, this right expression in (17) is called the *minimum-norm solution*.

Apart from the presented *minimum-norm solution*, there exists a general solution form for this LS problem that includes all optimal solutions despite of being non-minimal. This general solution form is described hereafter.

First, let us define the involved elements, *i.e.,* manifold subspaces and their respective subspace projectors, such as

- **Subspaces:**
  - Range of $\boldsymbol{A}$: $\mathcal{R}(\boldsymbol{A}) = \{\boldsymbol{y} \in \mathbb{R}^r | \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x} \in \mathbb{R}^j\}$ (18)
  - Null-space of $\boldsymbol{A}$: $\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{x} \in \mathbb{R}^j | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}$ (19)
- **Projectors:**
  - Range of $\boldsymbol{A}$: $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} = \boldsymbol{A}\boldsymbol{A}^+$ (20)
  - Null-space of $\boldsymbol{A}$: $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{A})} = \boldsymbol{I} - \boldsymbol{A}^+\boldsymbol{A}$ (21)

For the sake of simplicity, from now on the reference to $\boldsymbol{A}$ is omitted and hence $\boldsymbol{P}_{\mathcal{R}} = \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}$ and $\boldsymbol{P}_{\mathcal{N}} = \boldsymbol{P}_{\mathcal{N}(\boldsymbol{A})}$. With the aim of helping in the comprehension of these concepts, some of the elements stated above are graphically represented in Figure 3. We can see the $\boldsymbol{x} \in \mathbb{R}^j$ is linearly mapped onto $\mathbb{R}^r$ by means of $\boldsymbol{A}$. As the description above says, $\mathcal{R}(\boldsymbol{A})$ is formed by the resulting set of mapping all $\boldsymbol{x}$'s into $\mathbb{R}^j$ towards $\mathbb{R}^r$ using $\boldsymbol{A}$. Thus, $\boldsymbol{y} \in \mathbb{R}^r$ will also be inside $\mathcal{R}(\boldsymbol{A})$. As it is shown in the image, $\boldsymbol{P}_{\mathcal{N}}$ is a linear map from $\mathbb{R}^j$ to $\mathbb{R}^j$ and $\boldsymbol{P}_{\mathcal{R}}$ is a linear map from $\mathbb{R}^r$ to $\mathbb{R}^r$.

As noted before, $\boldsymbol{b}$ will not be exactly on $\mathcal{R}(\boldsymbol{A})$, hence $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$. However, if we project $\boldsymbol{b}$ onto $\mathcal{R}(\boldsymbol{A})$ using the projector $\boldsymbol{P}_{\mathcal{R}}$ described in (20), we can say that

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{A}^+\boldsymbol{b} \rightarrow \boldsymbol{A}\left(\boldsymbol{x} - \boldsymbol{A}^+\boldsymbol{b}\right) = \boldsymbol{0}\,. \tag{22}$$

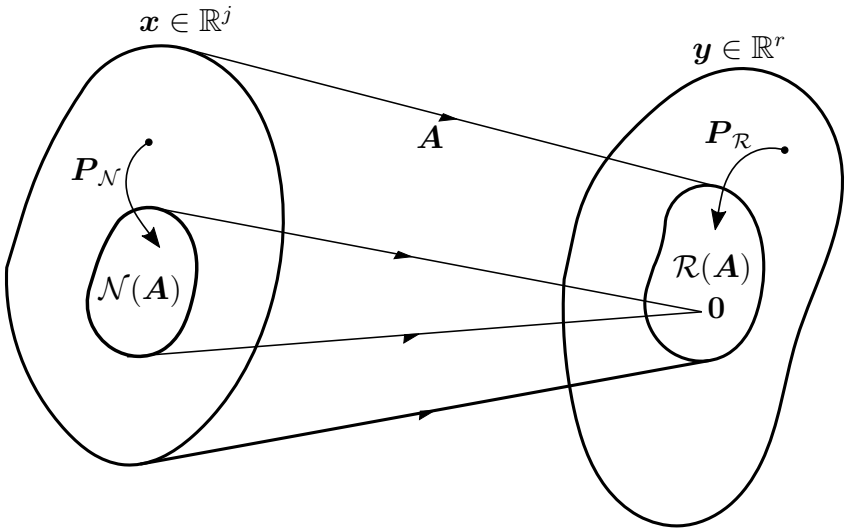


Figure 3: Spaces and subspaces involved in the general form of the LS solution.

From the *null-space* definition in (19), we see that the condition for a vector to belong to the null-space $\mathcal{N}(\boldsymbol{A})$ is that it should be mapped to $\mathbf{0}$ whenever is pre-multiplied by $\mathbf{A}$. Therefore, from (22) we can deduce that $\left(\boldsymbol{x} - \boldsymbol{A}^{+}\boldsymbol{b}\right) \in \mathcal{N}(\boldsymbol{A})$, and from this we derive the *general solution* form for the LS problem

$$\left(\boldsymbol{x} - \boldsymbol{A}^{+}\boldsymbol{b}\right) = \left(\boldsymbol{I} - \boldsymbol{A}^{+}\boldsymbol{A}\right)\boldsymbol{\nu} \rightarrow \boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b} + \left(\boldsymbol{I} - \boldsymbol{A}^{+}\boldsymbol{A}\right)\boldsymbol{\nu}\,, \tag{23}$$

being $\boldsymbol{\nu}$ an arbitrary vector. Notice how the form of (23) is the minimum-norm LS solution with an additional vector projected to $\mathcal{N}(\boldsymbol{A})$, *i.e.,* the second component modifies $\boldsymbol{x}$ but it does not modify the cost function $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ since it is projected onto the null-space.

Moving back to the inverse kinematic notation used in previous sections, the solution for (11) for a single task can be written as:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{+}\dot{\boldsymbol{\sigma}}_1 + \left(\boldsymbol{I} - \boldsymbol{J}^{+}\boldsymbol{J}\right)\dot{\boldsymbol{q}}_2 = \boldsymbol{J}^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1\,\dot{\boldsymbol{q}}_2\,, \tag{24}$$

where $\boldsymbol{P}_1$ is the null space projector onto the main task (1) and $\dot{\boldsymbol{q}}_2$ is a vector of joint velocities, which can be chosen wisely to perform a secondary task (2), without influencing the resolution of the main task (1). This numbering of variables has been introduced in (24) to help the reader identify the elements from each task.

LS with task hierarchy

Following with the concepts explained above, here we describe how to obtain the joint velocities solution when we have two or more tasks involved. Let us begin with two tasks which can be expressed in its differential form as:

$$\text{task 1 with priority 1: } \dot{\boldsymbol{\sigma}}_1 = \boldsymbol{J}_1 \dot{\boldsymbol{q}}\,, \tag{25a}$$

$$\text{task 2 with priority 2: } \dot{\boldsymbol{\sigma}}_2 = \boldsymbol{J}_2 \dot{\boldsymbol{q}}\,, \tag{25b}$$

where task 1 has a higher priority than task 2. In order to combine these two tasks, first we recall (24), such as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1\dot{\boldsymbol{q}}_2\,, \tag{26}$$

where $\boldsymbol{P}_1 = \left(\boldsymbol{I} - \boldsymbol{J}_1^{+}\boldsymbol{J}_1\right)$ is the projector to the null-space of task 1 and $\dot{\boldsymbol{q}}_2$ is the joint configuration only devoted to accomplish with task 2. The resulting joint velocities $\dot{\boldsymbol{q}}$ of (26) directly accomplish with task 1 ($\dot{\boldsymbol{\sigma}}_1$), but here we have to find $\dot{\boldsymbol{q}}_2$ so that task 2 can also be fulfilled if there exist enough DoF redundancy.

Introducing (26) in (25b), we obtain

$$\dot{\boldsymbol{\sigma}}_2 = \boldsymbol{J}_2\left(\boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1\dot{\boldsymbol{q}}_2\right)\,, \tag{27}$$

where $\dot{\boldsymbol{q}}_2$ can be isolated after few developments

$$\dot{\boldsymbol{q}}_2 = (\boldsymbol{J}_2\boldsymbol{P}_1)^{+}(\dot{\boldsymbol{\sigma}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1)\,. \tag{28}$$

If we substitute (28) back in (26), we get a final expression that hierarchically solves the IK problem for the tasks 1 and 2 expressed in (25a) and (25b)

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1(\boldsymbol{J}_2\boldsymbol{P}_1)^{+}(\dot{\boldsymbol{\sigma}}_2 - \boldsymbol{J}_2\boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1)\,. \tag{29}$$

This formulation can be extended to incorporate extra tasks with lower priority. For example, the incorporation of a third task into the general control law to obtain the joint velocities results in

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1\dot{\boldsymbol{q}}_2 + \boldsymbol{P}_2\dot{\boldsymbol{q}}_3\,, \tag{30}$$

where $\boldsymbol{P}_2 = \boldsymbol{P}_1\boldsymbol{P}_2^{'}$, with $\boldsymbol{P}_2^{'} = \boldsymbol{I} - (\boldsymbol{J}_2\boldsymbol{P}_1)^{+}(\boldsymbol{J}_2\boldsymbol{P}_1)$ the projector to the null-space of $\boldsymbol{J}_2\boldsymbol{P}_1$ –see (28)–; and $\dot{\boldsymbol{q}}_3$ is an arbitrary configuration that can be used for an eventual third task.

With the third task defined as $\dot{\boldsymbol{\sigma}}_3 = \boldsymbol{J}_3\dot{\boldsymbol{q}}$, and considering that the resulting joint velocities in (30) should also satisfy this last task, now we can obtain, with a similar procedure as to obtain (28) for the second task

$$\dot{\boldsymbol{q}}_3 = (\boldsymbol{J}_3\boldsymbol{P}_2)^{+}\left(\dot{\boldsymbol{\sigma}}_3 - \boldsymbol{J}_3\left(\boldsymbol{J}_1^{+}\dot{\boldsymbol{\sigma}}_1 + \boldsymbol{P}_1\dot{\boldsymbol{q}}_2\right)\right)\,. \tag{31}$$

Note how (30), combined with (28) and (31), has an iterative structure, exploited in [24] or [15], which allows us to provide a generic expression for n levels of hierarchy as follows:

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_p = \sum_{i=1}^{n} \boldsymbol{P}_{i-1}(\boldsymbol{J}_i\boldsymbol{P}_{i-1})^{+}(\dot{\boldsymbol{\sigma}}_i - \boldsymbol{J}_i\dot{\boldsymbol{q}}_{i-1})\,, \tag{32}$$

where $\boldsymbol{P}_0 = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$ and $\boldsymbol{P}_i = \boldsymbol{P}_{i-1}\boldsymbol{P}_i^{'}$ the projector into the null space of the task i constrained by the preceeding tasks.

2.4.2 Quadratic programming approach

QP problems, when applied in solving the IK problem, can be described as the process of solving a (linearly constrained) quadratic optimization problems for a set of tasks. That is, the problem of optimizing with respect to the joint variables a quadratic objective function of the tasks subject to some linear constraints.

As mentioned in Section 2.1, the LS approach has the advantage of providing an analytical solution. When implementing algorithms that are running onboard on real robots, we have to ensure that the algorithm burden allows for an online implementation. Computing an analytical solution is faster than solving an optimization problem. However, recent improvements on computer computation capacity and the existence of specific optimized solvers let us implement algorithms based on optimization problems that do not have any analytical solution.

The main reason to implement a QP approach (or generally implementing an optimization-based approach without an analytical solution) is the possibility to include inequality tasks as problem constraints. When dealing with equality tasks one must ensure that the configuration found $\dot{\boldsymbol{q}}$ satisfies a task specific value $\boldsymbol{J}\dot{\boldsymbol{q}} = \dot{\boldsymbol{\sigma}}$. However, with inequality tasks it is enough to ensure that the solution $\dot{\boldsymbol{q}}$ remains in the subspace where the following inequality is still accomplished:

$$\boldsymbol{J}\dot{\boldsymbol{q}} \lessgtr \dot{\boldsymbol{\sigma}}\,. \tag{33}$$

Similarly to Section 2.4.1, in this section we describe the fundamentals of QP problems with a generic mathematical nomenclature, which will be re-adapted at the end of the section.

Figure 4 shows a simple scheme with a 2D space ($\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top$), where we can see how the feasible set for an inequality task (green area) is wider than the one for an equality task (over the brown line). The optimal solution for an inequality task belongs to a region

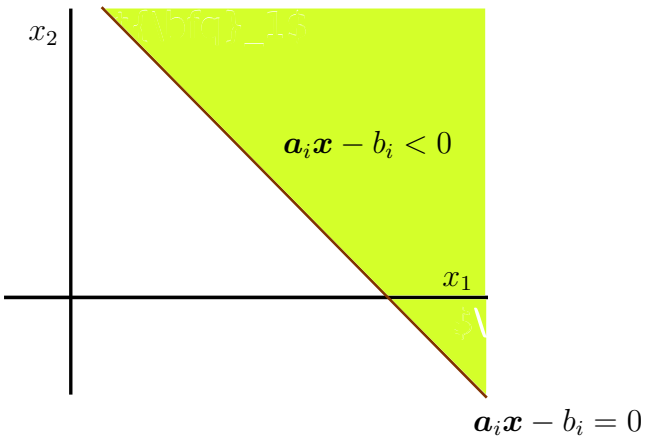


Figure 4: Equality and inequality tasks representation for a 2-dimensional space.

whose dimension is the same of its ambient space -in the case of Figure 4, this dimension is 2-. This region is bounded by a hyperplane[1](a line in 2D). In case the task is an equality task, the optimal solution resides in a hyperplane. Some examples for inequality tasks might be to add bounds to the joint values or even to perform obstacle avoidance tasks by keeping safe collision distances.

Optimization problems with a QP form, consider quadratic objective functions with affine constraints. These constraints are hardly imposed over the problem variables during the iterative solution of the optimization problem. Although they can be set to limit or bound the problem variables (*i.e.,* task variables), hereafter we only specify those constraints directly related with imposing the hierarchy between tasks. These bounds will be set according to the experimental case studies in their respective sections.

This type of problems has the following generic form:

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} . & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{l}^\top \boldsymbol{x} + u \\ \text{s.t.} & \boldsymbol{C}\boldsymbol{x} \preceq \boldsymbol{d} \\ & \boldsymbol{E}\boldsymbol{x} = \boldsymbol{g}\,, \end{array} \tag{34}$$

where $\boldsymbol{H} = 2\boldsymbol{A}^\top\boldsymbol{A}$, $\boldsymbol{l}^\top = -2\boldsymbol{b}^\top\boldsymbol{A}$ and $u = \boldsymbol{b}^\top\boldsymbol{b}$ from (34). The matrix form $\boldsymbol{C}\boldsymbol{x} \preceq \boldsymbol{d}$ is a compact manner to express several inequality constraints ($\preceq$ states for component-wise vector inequality and also equality constraints). The number of rows in $\boldsymbol{C}$ indicates how many individual inequality constraints the problem has (an individual inequality constraint is shown in Figure 4). Analogously, several equality constraints are expressed in their matrix form with the expression $\boldsymbol{E}\boldsymbol{x} = \boldsymbol{g}$.

[1]A hyperplane is a subspace whose dimension is one less than that of its ambient space.

Notice how the least squares formulation is, in fact, a specific case of quadratic programming. Actually, sometimes we can refer to it as unbounded QP problem. Hence, the least squares equation (11) can be easily expanded so it takes the form in (34)

$$\begin{aligned} ||\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}||_2^2 &= (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^\top (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \\ &= \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{b}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{b}\,. \end{aligned} \tag{35}$$

One of the challenges of using a QP approach is to find a way to specify task hierarchies. In the following we describe 2 methods to impose this task hierarchy.

QP problem with task pseudo-hierarchy

The first approach to specify a hierarchy between cost functions (*i.e.,* tasks) is actually not considering a strict hierarchy but a weighted sum of all tasks. In this way, the weights are set to determine the importance of each task, thus assigning a higher weight to those tasks that we consider more important. Hence, the problem will look like the following:

$$\begin{aligned} \min_{\boldsymbol{x}}\,. \quad & \sum_{i=1}^{n} \tfrac{w_i}{h_i} \left(\boldsymbol{x}^\top \boldsymbol{H}_i \boldsymbol{x} + \boldsymbol{l}_i^\top \boldsymbol{x}\right) \\ \text{s.t.} \quad & \boldsymbol{C}\boldsymbol{x} \preceq \boldsymbol{d} \\ & \boldsymbol{E}\boldsymbol{x} = \boldsymbol{g}\,, \end{aligned} \tag{36}$$

where w_i is the associated weight of the task i and h_i is a normalization factor that ease the choice of the weights. When a weighted sum is used, it is important to normalize the objective functions, in order to effectively set the desired weights w_i. Thus, we chose the factors h_i equal to the spectral norm of $\boldsymbol{H}_i$, which is equal to the square root of the largest eigenvalue of the product matrix $\boldsymbol{H}_i^\top \boldsymbol{H}_i$ [16]. Note that the scalar u of the cost function has disappeared. This is because this factor is just a bias in the cost function and its minimum does not depend on u. Therefore, the value of $\boldsymbol{x}$ that minimizes the cost function is the same in both cases -with and without the scalar u-.

In order to show how to express (36) with the IK notation, we will consider two equality tasks and two inequality tasks, *i.e.,*

$$\begin{aligned} \dot{\boldsymbol{\sigma}}_i &= \boldsymbol{J}_i(\boldsymbol{q})\,\dot{\boldsymbol{q}}, \quad i = 1, 2\,, \\ \dot{\boldsymbol{\sigma}}_i &\geq \boldsymbol{J}_i(\boldsymbol{q})\,\dot{\boldsymbol{q}} \quad i = 3, 4\,. \end{aligned} \tag{37}$$

The equivalent QP problem with pseudo-hierarchy would be of the form

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}}\,. \quad & \sum_{i=1}^{2} \tfrac{w_i}{h_i} \left(\dot{\boldsymbol{q}}^\top \boldsymbol{J}_i^\top \boldsymbol{J}_i \dot{\boldsymbol{q}} - 2\dot{\boldsymbol{\sigma}}_i^\top \boldsymbol{J}_i \dot{\boldsymbol{q}}\right) \\ \text{s.t.} \quad & \begin{bmatrix} \boldsymbol{J}_3 \\ \boldsymbol{J}_4 \end{bmatrix} \dot{\boldsymbol{q}} \preceq \begin{bmatrix} \boldsymbol{\sigma}_3 \\ \boldsymbol{\sigma}_4 \end{bmatrix}\,. \end{aligned} \tag{38}$$

In the case that we want task 1 to have priority over task 2, we should set $w_1 > w_2$. The solution for this problem, however, will lead in some cases to a trade off between task configurations, *i.e.,* when two task require antagonistic configurations we might not fulfill completely any of the tasks. Besides, there is no way of saying which inequality task is more important since all of them are implemented here as constraints in the quadratic programming problem.

QP problem with strict task hierarchy

In Section 2.4.1, we explained how to find an analytical solution for the LS approach imposing a strict hierarchy among tasks. Furthermore, we have seen that a LS problem is a specific case of a QP problem. Thus, it seems reasonable that the LS problem with strict hierarchy has also its equivalent QP hierarchical problem.

As this formulation will be the base of our work applied to robotics fields, the notation of the following sections is expressed in terms of the IK notation instead of the generic notation used so far.

Let us consider again two task-derivatives with their corresponding Jacobians, as in (25). We want, if possible, to accomplish both tasks with the robot and, in case of impossibility, to accomplish the task with a higher priority, *i.e.,* task 1. Therefore, we first have to solve the IK problem for the first task using the QP approach as the following optimization problem:

$$\min_{\dot{\boldsymbol{q}}} . \quad \dot{\boldsymbol{q}}^\top \boldsymbol{J}_1^\top \boldsymbol{J}_1 \dot{\boldsymbol{q}} - 2\dot{\boldsymbol{\sigma}}_1^\top \boldsymbol{J}_1 \dot{\boldsymbol{q}} \,. \tag{39}$$

Solving this minimization problem results in obtaining a vector of joint velocities that guarantees the minimization of task 1 (*i.e.,* accomplishment of task 1), namely $\dot{\boldsymbol{q}}^1$. Notice how the superscript $\dot{\boldsymbol{q}}^a$ specifies from which QP problem is solution the variable, *e.g.,* $\dot{\boldsymbol{q}}^1$ is the joint velocities vector obtained from the first optimization.

Now it is time to solve the problem for task 2 ensuring that the solution found still accomplishes the task 1. This is done by running a second optimization procedure with task 2, but this time adding the following equality constraint

$$\boldsymbol{J}_1 \dot{\boldsymbol{q}} = \boldsymbol{J}_1 \dot{\boldsymbol{q}}^1 \,. \tag{40}$$

Therefore, the final solution for the two-tasks-hierarchical problem comes up from solving

also the following QP problem:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} . \quad & \dot{\boldsymbol{q}}^\top \boldsymbol{J}_2^\top \boldsymbol{J}_2 \dot{\boldsymbol{q}} - 2\dot{\boldsymbol{\sigma}}_2^\top \boldsymbol{J}_2 \dot{\boldsymbol{q}} \\ \text{s.t.} \quad & \boldsymbol{J}_1 \dot{\boldsymbol{q}} = \boldsymbol{J}_1 \dot{\boldsymbol{q}}^1 . \end{aligned} \tag{41}$$

The procedure explained above with two tasks and two nested optimization problems can be extended to a problem with n hierarchical tasks and n QP problems to get the final value for $\dot{\boldsymbol{q}}^n$ (lexicographic approach). This is an equivalent procedure to the one explained in 2.4.1. Although this scheme of nested optimizations has a clear drawback in terms of computation time, it allows us to specify useful inequality tasks and constraints.

Adding inequality tasks

In the QP with *pseudo-hierarchy* approach, we have shown how to deal with inequality tasks. There, the inequality tasks were added as hard constraints of the single QP problem. These inequality tasks had no hierarchy among them and, furthermore, introducing more than one inequality task could lead to an empty feasible set which results in an unfeasible QP problem.

In the following, we present a method that will allow us to avoid the unfeasibility issue and prioritize among inequality tasks [9]. This unfeasibility issue is solved by adding slack variables into the inequalities and, similarly to the case of equality tasks, the hierarchy among inequalities is performed solving nested QP problems.

To better explain the methodology, let us use an example. Consider the following three inequality tasks with priority 1 (the highest) and one equality task with a lower priority:

$$\begin{aligned} \dot{\boldsymbol{\sigma}}_i &\geq \boldsymbol{J}_i(\boldsymbol{q})\, \dot{\boldsymbol{q}}, \quad i = 1, ..., 3 \quad && (\textit{priority } 1)\,, \\ \dot{\boldsymbol{\sigma}}_4 &= \boldsymbol{J}_4(\boldsymbol{q})\, \dot{\boldsymbol{q}} && (\textit{priority } 2)\,. \end{aligned} \tag{42}$$

A slack variable must be added for each inequality task defined as

$$\boldsymbol{w}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\, \dot{\boldsymbol{q}} - \dot{\boldsymbol{\sigma}}_i\,, \tag{43}$$

being $\boldsymbol{w}_i \in \mathbb{R}_+^{m_i}$ the slack variable with m the dimension of the *i-th* task space. Notice how the slack variable will always be positive definite (positive orthant), because if $\dot{\boldsymbol{\sigma}}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\, \dot{\boldsymbol{q}}$ then the task i would already be accomplished and the minimization process finished[2].

In order to find a feasible solution accomplishing the three tasks, the QP problem is now

[2]Further details on *slack variables* are in Appendix A.1.

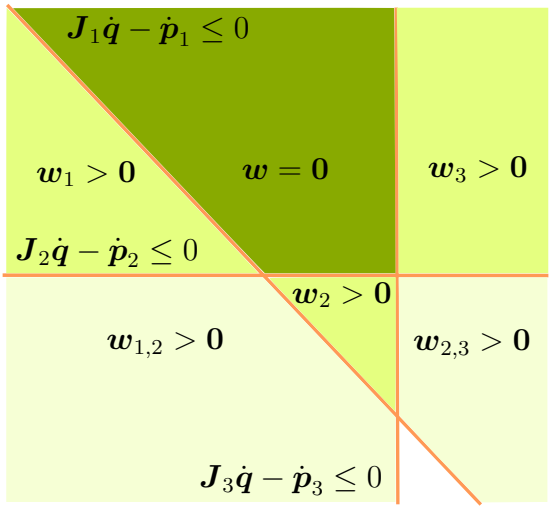


Figure 5: Representation of a possible feasible region bounded by 3 inequality constraints. A darker color indicates that mores inequalities are fulfilled.

expressed as

$$\begin{aligned} \min_{\dot{\boldsymbol{q}},\boldsymbol{w}} \; & \tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} \\ \text{s.t.} \; & \boldsymbol{w}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{\sigma}}_i, \quad i = 1, ..., 3\,, \end{aligned} \tag{44}$$

being $\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \boldsymbol{w}_3 \end{bmatrix}^\top$. Here we optimize the values of $\dot{\boldsymbol{q}}$ and $\boldsymbol{w}$. From (43), we see how the better the $\dot{\boldsymbol{q}}$ values to accomplish the tasks the minimal $\boldsymbol{w}$. In case of having perfectly feasible tasks (*i.e.,* all tasks can be accomplished simultaneously), we will reach $\boldsymbol{w} = \boldsymbol{0}$. Thus in the optimal problem we look for the minimal values of $\boldsymbol{w}$. However we cannot let the optimization process minimize $\boldsymbol{w}_i$ if the *i-th* tasks cannot be accomplised, therefore we introduce the accomplishment of the task as a constraint in the optimization process. In this way, the optimization results in values of $\dot{\boldsymbol{q}}$ that satisfy all task inequalities eventhough some slack variables cannot be minimized to $\boldsymbol{0}$. Solving this first QP problem will result in obtaining the optimal values $\boldsymbol{w}^1$ and $\dot{\boldsymbol{q}}^1$ that satisfy the inequality tasks with priority 1.

Figure 5 shows a simple example of a 2D space with all regions involved having 3 inequality tasks. Here, there exists a region where all three tasks are fulfilled (darker green) and therefore, inside this region $\boldsymbol{w} = \boldsymbol{0}$ (no constraint is violated). Notice that any value for $\dot{\boldsymbol{q}}$ that lies inside the darker green is an optimal solution for the problem stated at (44).

The task $\dot{\boldsymbol{\sigma}}_4$ from (42), which has lower priority, will be solved as another QP problem similarly to (39). In order to preserve the hierarchy of the first tasks (priority 1), we add the inequality tasks of the first optimization as hard constraints, *i.e.,* without the slack

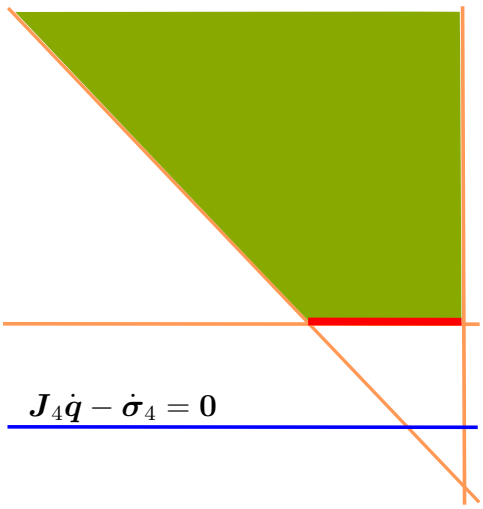


Figure 6: Representation of the solution sets for all tasks. Inequality task spaces as in Figure 5 together with the solution set for the fourth task represented with a blue line (equality task with priority 2) in case of an unconstrained problem. The final constrained solution that fulfills the hierarchy imposed is drawn as a red line, showing how the inequality tasks are imposed over the equality one, which cannot be fulfilled unless it crosses the dark green area.

variables as shown in the following optimization problem:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} . \quad & \dot{\boldsymbol{q}}^\top \boldsymbol{J}_4^\top \boldsymbol{J}_4 \dot{\boldsymbol{q}} - 2\dot{\boldsymbol{\sigma}}_4^\top \boldsymbol{J}_4 \dot{\boldsymbol{q}} \\ \text{s.t.} \quad & \dot{\boldsymbol{\sigma}}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\dot{\boldsymbol{q}}, \qquad i = 1, ..., 3\,, \end{aligned} \tag{45}$$

where we would obtain joint velocities $\dot{\boldsymbol{q}}^2$ which are the best ones to try to accomplish the task with priority 2 ($\dot{\boldsymbol{\sigma}}_4$) respecting the accomplishment of the tasks with higher priority. Later in this section, we describe the special case where an inequality constraint has been violated, which can be trated as an equality in subsequent QP problems.

Figure 6 shows the solution sets over the same example space as in Figure 5, but this time incorporating the optimal set for the second optimization problem, stated in (45). The blue line represents the set of solutions for the fourth task in the event of an unconstrained problem. However, in this problem statement we imposed a hierarchy among tasks, where the inequalities have priority over the equality task. We can see that the intersection of the dark green region and the blue line is a void set, meaning that all tasks cannot be fulfilled at the same time. Therefore, the optimization problem (45) will minimize the error for the fourth task $||\boldsymbol{J}_4\dot{\boldsymbol{q}} - \dot{\boldsymbol{\sigma}}_4||^2$ considering only values inside the dark green region,

leading to the solution set represented by a red line in Figure 6.

In the previous lines, we explained how to proceed in the event of inequality tasks having higher priority than equality task. Let us now consider a change in the priorization, setting the equality task with the highest priority, as follows:

$$\begin{array}{lll} \dot{\boldsymbol{\sigma}}_1 = \boldsymbol{J}_1(\boldsymbol{q})\,\dot{\boldsymbol{q}} & & (\textit{priority } 1)\,, \\ \dot{\boldsymbol{\sigma}}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\,\dot{\boldsymbol{q}}, & i = 2, ..., 4 & (\textit{priority } 2)\,. \end{array} \tag{46}$$

In this case, we would solve first an analogous problem to (39), getting an optimal solution expressed as $\dot{\boldsymbol{q}}^1$. In the second optimization problem, where the inequality tasks are introduced, the slack variables would play a role again, with the following QP problem:

$$\begin{array}{ll} \underset{\dot{\boldsymbol{q}},\boldsymbol{w}}{\min}. & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} \\ \text{s.t.} & \boldsymbol{w}_i \geq \boldsymbol{J}_i(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{\sigma}}_i, \quad i = 2, ..., 4 \\ & \boldsymbol{J}_1\dot{\boldsymbol{q}} = \boldsymbol{J}_1\dot{\boldsymbol{q}}^1\,. \end{array} \tag{47}$$

The solution for this second QP problem would result in $\boldsymbol{w}^2$ and the velocities of the joints fulfillung the inequalitiy tasks with priority 2, $\dot{\boldsymbol{q}}^2$. The graphical representation for this problem example is shown in Figure 7. The final solution set is represented by the red line. In contrast to the previous hierarchy situation, here the solution lies in the equality task because it has a higher priority. Minimizing the slack variables in (47) ensures that the solution fulfills the inequalities as much as possible, *i.e.,* the red line in Figure 7 fulfills 2 out of the 3 existing inequality constraints. Thus, the slack variable associated to inequality tasks 1 and 3, $\boldsymbol{w}_1$ and $\boldsymbol{w}_3$, will be $\mathbf{0}$ while the one associated with task 2, $\boldsymbol{w}_2$, will be grater than $\mathbf{0}$ (strictly positive).

The addition of extra equality or inequality tasks will follow similar procedures as the ones described above with nested optimization problems. For example, if we add yet another task ($\dot{\boldsymbol{\sigma}}_5 \lessgtr \boldsymbol{J}_5(\boldsymbol{q})\,\dot{\boldsymbol{q}}$) with an eventual priority 3, we would run a third QP problem proceeding depending if $\dot{\boldsymbol{\sigma}}_5$ is an equality or inequality task, such as (39) or (44) respectively, but this time incorporating as constraints the tasks higher in the hierarchy. With the priorization of this last example, described in (46), neither the first task nor the third are violated, then they must added as inequality tasks following the same procedure as in (45). However, the second inequality is violated and therefore it can be directly added as an equality constraint using the expression $\boldsymbol{J}_2\dot{\boldsymbol{q}} = \boldsymbol{J}_2\dot{\boldsymbol{q}}^2$. In this way, we force the solution to the third QP problem to still keep the minimum distance to the inequality task 2.

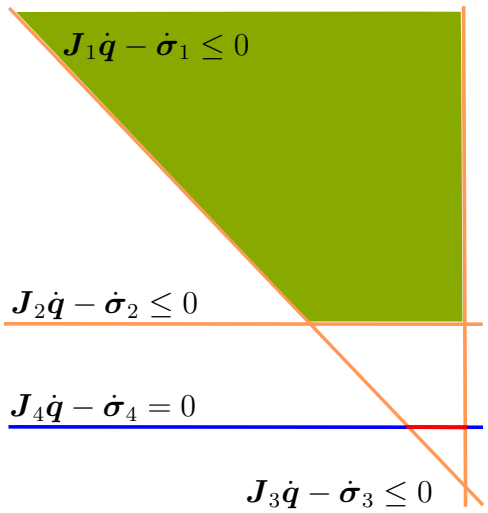


Figure 7: Representation of the equality and inequalities task solution sets. Here the equality task has priority over the inequalities, leading to a solution that fulfills the hierarchy imposed over the red line.

2.5 Closed-loop inverse kinematics

In the formulation presented so far, we have assumed the operation of the robot to be ideal. The IK assumes that the geometry is perfect and that no perturbation affects the evolution of a task, which is known as the pure IK problem. However, this idealization does not reflect the real situation where the robot geometry is not perfect and several perturbations might not be mathematically considered.

In order to deal with this not predicted events and still accomplish with the required tasks, the CLIK closes a control loop for the IK problem by comparing the current and desired values of a task, obtaining an expression of the task error such as

$$\boldsymbol{e}_i(t) = \boldsymbol{\sigma}_i^d(t) - \boldsymbol{\sigma}_i(t)\,, \tag{48}$$

being $\boldsymbol{e}_i(t)$ the task error for the *i-th* task at time t. $\boldsymbol{\sigma}_i^d(t)$ and $\boldsymbol{\sigma}_i(t)$ are the desired and current *i-th* task values, respectively. All these variables exist in $\mathbb{R}^r$, being r the dimension of the task-space. In the following developments we avoid the mention to time t and task number i for the sake of clarity and conciseness.

Our objective is to reduce this error $\boldsymbol{e}$ towards $\mathbf{0}$, hence we can impose a dynamics on this error to force this convergence with

$$\dot{\boldsymbol{e}} = -\lambda \boldsymbol{e}\,, \tag{49}$$

being $\dot{\boldsymbol{e}} \in \mathbb{R}^r$ the time-derivative of the error and λ a weight respecting $\lambda > 0$. Notice how λ is a scalar, therefore the same dynamics is assigned to every component of $\boldsymbol{e}$. If we want to apply different dynamics, we might consider the use of a diagonal matrix such that

$$\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_r) \succ 0\,, \tag{50}$$

where "$\succ$" stands for a positive definite matrix.

Expanding equation (49) and combining it with the generic task formulation from (1) and (2), we get

$$\boldsymbol{J}\dot{\boldsymbol{q}} - \left(\dot{\boldsymbol{\sigma}}^d + \lambda(\boldsymbol{\sigma}^d - \boldsymbol{\sigma})\right) = \boldsymbol{0}\,. \tag{51}$$

Solving the CLIK problem means to find the values of $\dot{\boldsymbol{q}}$ so (51) holds. This problem can be solved with any method presented before as a mathematical optimization problem (LS or QP). For example, assimilating (51) with the general form of a LS cost function as in (12), we would have

$$\min_{\dot{\boldsymbol{q}}} ||\boldsymbol{J}\dot{\boldsymbol{q}} - \left(\dot{\boldsymbol{\sigma}}^d + \lambda(\boldsymbol{\sigma}^d - \boldsymbol{\sigma})\right)||_2^2\,. \tag{52}$$

For the QP case, all concepts explained in section 2.4.2 hold. We just have to consider the CLIK task to be

$$\boldsymbol{J}\dot{\boldsymbol{q}} \lessgtr \widetilde{\dot{\boldsymbol{\sigma}}}\,, \tag{53}$$

where in this case, $\widetilde{\dot{\boldsymbol{\sigma}}} = \left(\dot{\boldsymbol{\sigma}}^d + \lambda(\boldsymbol{\sigma}^d - \boldsymbol{\sigma})\right)$.

2.5.1 Task stability

Closing the loop of the IK problem (*i.e.,* CLIK) arises a basic question related to the error performance: is this error stable (decreases with the time)? We consider this discussion out of the thesis scope, however in this section we want to point a basic concept to proof the stability of a task and therefore of the system. For further details, we refer the reader to [19], [20] or [21].

In Section 2.5, we described how the dynamics of the error is imposed with (49). In such a system, if we want the error to decrease towards $\boldsymbol{0}$, it is enough by setting $\lambda > 0$. However, if we expand (49) to (51) we can see how the equality will be fulfilled depending on the values of $\dot{\boldsymbol{q}}$.

Therefore, when solving a task by means of an optimization procedure, the error of this task will tend to $\boldsymbol{0}$ only if the solution found makes (51) to hold. This means that we assume that a global and exact solution of the problem for each task is possible in a limited computational time (*i.e.,* here feasibility implies stability [25]).

When combining tasks within a hierarchical control law, the global system stability is kept if the tasks are stable. Even though the errors of the tasks with lowest priority will be more difficult to reduce (or could even not reach $\mathbf{0}$ values if the tasks higher in the hierarchy do not leave enough free DoFs to accomplish them), their errors will still tend to $\mathbf{0}$ or to the smallest possible values. It is important to consider the use of the robot DoFs while designing the mission tasks, because the feasible set is reduced as constraints (related with higher-priority tasks) are introduced in the optimization problem. Thus the null-space of the main task becomes smaller, suggesting that even though the system might be stable, not all tasks might be accomplished. Hence, it is interesting to choose a strategy to set the tasks stack, prioritizing the tasks related with the platform integrity and stability (*e.g.,* collision avoidance).

2.5.2 Task expression in the acceleration domain

So far, we have defined the task control law working in the velocity domain, *i.e.,* $\dot{\boldsymbol{\sigma}}_k$ as in (2), (25), (37), (42), (46); leading to the error definition in (48) with the imposed dynamics on the derivative for the CLIK problem. We kept these definitions for the sake of comprehension as it is commonly done in the literature, however generating the trajectory in the acceleration domain will give us some advantages.

Even though we are only performing a kinematic analysis, formulating the tasks considering accelerations will contribute in a better performance of the robot with smoother trajectories in the position domain. Hence, we can choose a second order linear system to model the closed-loop characteristics of the error. Therefore, the dynamics of the error can be expressed as

$$\ddot{\boldsymbol{e}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} = \mathbf{0}\,, \tag{54}$$

where $\ddot{\boldsymbol{e}} \in \mathbb{R}^r$ and $\dot{\boldsymbol{e}} \in \mathbb{R}^r$ are acceleration and velocity errors respectively. $\boldsymbol{K}_D \in \mathbb{R}^{r\times r}$ and $\boldsymbol{K}_P \in \mathbb{R}^{r\times r}$ are gain matrices, typically diagonal. Recall that in this case, r is the dimension of the task space.

By choosing properly the values of the matrices $\boldsymbol{K}_D$ and $\boldsymbol{K}_P$, we could ensure the *asymptotic stability* of the error, *i.e.,* the error tends to $\mathbf{0}$ as time tends to infinity. In the following we present a procedure to define those matrices by taking advantage of the Gronwall's inequality.

The Gronwall's inequality is a sufficient condition which ensures that a function expressed as $f \leq 0$ is satisfied if $\dot{f} \leq -\lambda f$, where λ is a positive scalar gain. Here we apply the Gronwall's inequality to an equality task and, in order to ease the comprehension, we consider a task with dimension 1 and, therefore, gain matrices in equation (54) will now

be scalars k_p and k_d.

First, we want to ensure the convergence of the first order system towards 0 by imposing

$$\dot{e} = -\lambda_1 e\,, \tag{55}$$

being λ_1 a scalar bigger than 0, which is equivalent as placing a pole at $-\lambda_1$ of the transfer function in traditional control. Now, the result in (55) is assigned to a new function $\overline{e} = \dot{e} + \lambda_1 e$. Proceeding again as in (55) with this new function $\overline{e}$ we obtain

$$\dot{\overline{e}} = -\lambda_2 \overline{e}\,, \tag{56}$$

which can be expanded as

$$\ddot{e} + (\lambda_1 + \lambda_2)\dot{e} + \lambda_1\lambda_2 e = 0\,. \tag{57}$$

Therefore, value $k_D = (\lambda_1 + \lambda_2)$ and $k_P = \lambda_1\lambda_2$. We can express this second order system in its state-space form as

$$\begin{bmatrix} \dot{e} \\ \ddot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\lambda_1\lambda_2 & -(\lambda_1 + \lambda_2) \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix}. \tag{58}$$

Computing the eigenvalues of the state matrix in (58) we see that its eigenvalues are $-\lambda_1$ and $-\lambda_2$ with $\lambda_1 > 0$ and $\lambda_2 > 0$ by definition. Hence, we can conclude that the error dynamics is asymptotically stable.

Now, we want to express this error in terms of joint variables, task variables and their corresponding derivatives. Differentiating (2) with respect to time, we obtain

$$\ddot{\boldsymbol{\sigma}} = \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\ddot{\boldsymbol{q}}\,, \tag{59}$$

where $\ddot{\boldsymbol{\sigma}} \in \mathbb{R}^r$ are task accelerations, $\dot{\boldsymbol{q}} \in \mathbb{R}^j$ and $\ddot{\boldsymbol{q}} \in \mathbb{R}^j$ are the joint velocities and accelerations, respectively. $\dot{\boldsymbol{J}} \in \mathbb{R}^{r\times j}$ is the Jacobian time derivative (we omitted the dependence of the Jacobian on $\boldsymbol{q}$ for the sake of conciseness).

Then, we can substitute (59) in (57), getting the expression

$$\begin{aligned} (\ddot{\boldsymbol{\sigma}}_d - \ddot{\boldsymbol{\sigma}}) + (\lambda_1 + \lambda_2)(\dot{\boldsymbol{\sigma}}_d - \dot{\boldsymbol{\sigma}}) + \lambda_1\lambda_2(\boldsymbol{\sigma}_d - \boldsymbol{\sigma}) = 0\,, \\ (\ddot{\boldsymbol{\sigma}}_d - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \boldsymbol{J}\ddot{\boldsymbol{q}}) + (\lambda_1 + \lambda_2)(\dot{\boldsymbol{\sigma}}_d - \boldsymbol{J}\dot{\boldsymbol{q}}) + \lambda_1\lambda_2(\boldsymbol{\sigma}_d - \boldsymbol{\sigma}) = 0\,, \end{aligned} \tag{60}$$

where $\ddot{\boldsymbol{\sigma}}_d \in \mathbb{R}^r$ and $\dot{\boldsymbol{\sigma}}_d \in \mathbb{R}^r$ are the desired task accelerations and velocities, respectively. Note how with λ_i scalar variables, the same dynamics are assigned to each coordinate of the task error, in case of preferring different dynamics we should consider gain diagonal matrices instead of a scalar, as in (54).

In oder introduce the tasks in an SDP problem, equation (60) must be rearranged to the form of a QP cost function in the acceleration domain, $min||\boldsymbol{J}\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}||^2$. Hence, now the optimization variable is $\ddot{\boldsymbol{q}}$, and $\widetilde{\ddot{\boldsymbol{\sigma}}}$ can be extracted from the rest of (60), such as

$$\widetilde{\ddot{\boldsymbol{\sigma}}} = (\ddot{\boldsymbol{\sigma}}_d - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) + (\lambda_1 + \lambda_2)(\dot{\boldsymbol{\sigma}}_d - \boldsymbol{J}\dot{\boldsymbol{q}}) + \lambda_1\lambda_2(\boldsymbol{\sigma}_d - \boldsymbol{\sigma})\,. \tag{61}$$

To conclude this section, it is important to point out the importance of equations (60) and (61). They are a general expression of any kind of task in the acceleration domain. Furthermore, they allow us to specify, a part from the position reference, also references for the task velocities and accelerations. The most common way to proceed is to set a reference for the position while references for velocity and acceleration are zero. However, we might want to specify an specific profile for the task. This will imply the reference task position, velocity and acceleration to change with respect to the time and take values different from 0.

2.5.3 Discrete implementation[3]

All concepts explained in this work are developed in the continuous domain as it offers an easy to follow development of the formulation. However, all these concepts are related to computer science fields where processor units are the center of decisions. Therefore, we have to find the procedures to move the equations from continuous to discrete descriptions. In this section we describe the procedure of moving our formulation from the continuous domain to the discrete domain. This procedure has been used in order to implement the proof of concept of our developments described in Appendix B and also to perform the experiments with the real robot described in Chapter 4.

Considering a sufficiently small sampling time δt, (59) at time $t_k = k\delta t$ can be written as

$$\ddot{\boldsymbol{\sigma}}_k = \dot{\boldsymbol{J}}_k\dot{\boldsymbol{q}}_k + \boldsymbol{J}_k\ddot{\boldsymbol{q}}_k\,, \tag{62}$$

where $k = 0, 1, \ldots$ denotes the sampling of the corresponding measure.

The aim of all methods presented in this work is to solve the IK problem to find $\ddot{\boldsymbol{q}}_k$ in (62) to actuate our robot and accomplish the required task. $\ddot{\boldsymbol{q}}_k$ can be integrated if our robot is actuated through joint velocities or positions. Therefore, velocity and position references would have the following expressions

$$\begin{aligned} \dot{\boldsymbol{q}}_k &= \dot{\boldsymbol{q}}_{k-1} + \ddot{\boldsymbol{q}}_k\delta t\,, \\ \boldsymbol{q}_k &= \boldsymbol{q}_{k-1} + \dot{\boldsymbol{q}}_{k-1}\delta t + \ddot{\boldsymbol{q}}_k\frac{\delta t^2}{2}\,. \end{aligned} \tag{63}$$

[3]See the Section 1.3 for a discussion about the notation used in this section.

However, at every time step we still have to evaluate the other variables in (62). That is the task acceleration, the task Jacobian and its derivative as well as the velocity of the joints. Assuming we can take measurements up to the velocity level, the following approximations for the task error can be done

$$\ddot{\boldsymbol{\sigma}}_k \simeq \frac{\dot{\boldsymbol{\sigma}}_k - \dot{\boldsymbol{\sigma}}_{k-1}}{T}. \tag{64}$$

The time-derivative of the task Jacobian can be approximated with

$$\dot{\boldsymbol{J}}_k \simeq \frac{\boldsymbol{J}_k - \boldsymbol{J}_{k-1}}{T}, \tag{65}$$

where Jacobians depend only on the joints positions of the corresponing sample, *i.e.,* $\boldsymbol{q}_k$ and $\boldsymbol{q}_{k-1}$. Regarding to the joint velocity we have two options. We can either use previous reference calculated using (63) or we could use current and previous joint position measurements. According to this we would use any of the following:

$$\begin{aligned} \dot{\boldsymbol{q}}_k &\simeq \dot{\boldsymbol{q}}_{k-1}, \\ \dot{\boldsymbol{q}}_k &\simeq \frac{\boldsymbol{q}_k - \boldsymbol{q}_{k-1}}{T}. \end{aligned} \tag{66}$$

These discretizations are approximations that, although being close to the real values, they are not exact. However, as we are closing the loop (CLIK), the control law will compensate these inaccuracies. When implementing discrete systems, things such as the sampling period δt affect considerably to its stability. As pointed before, we consider the exhaustive study of stability out of the scope of this work. Stability in these kind of discrete systems has been the matter of study of several works such as [21].

2.6 Summary

In this chapter, we reviewed the stat-of-the-art on trajectory generation for highly redundant robots. Afterwards, we distinguished among kinematic and functional redundancies, based on the dimensions of the different spaces we are working on (*i.e.,* operational, joints and task spaces). That, led us to explain what a task is, with its formulation where the task Jacobian matrix has a capital role.

With the task concept clear, we presented different methods for solving the IK problem, including how to deal with different tasks simultaneously by applying a hierarchy among them. To do so, first we presented an analytical solution by means of a LS approach, then considering its limitations we moved to describe how to hierarchically combine tasks as QP problems.

The chapter ends with the explanation on how to use a CLIK strategy either in the velocity and acceleration domain along with some dissertations regarding its stability and its implementation in discrete time.

3. Hierarchical task control through lexicographic SDP

In the previous chapter we recalled the theoretical knowledge upon which we base our approach. Here, we present the formulation that brace the core of this thesis by proposing an algorithm to control a robot with prioritized tasks (hierarchy) through SDP lexicographic optimization.

First, we define the formulation and concepts to introduce a single task problem as an SDP. We offer a detailed explanation about the necessary transformations that suffer either the objective function (task) and its constraints. Then, we introduce the concept of hierarchy with SDPs together with the presentation of the main trajectory generation algorithm. Once again, we refer the reader to Appendix B for a didactic example of the concepts explained in this chapter.

3.1 Task resolution through semidefinite programming

In this section we adapt the task equations stated previously. The proposed formulation by means of *Linear Matrix Inequalities* (LMIs), allow us to state a mathematical optimization problem that can be solved in real time using dedicated solvers. Once again, we start the chapter with a generic mathematical notation and specialize it to the IK notation for the particular developments at the end.

A *Semidefinite Programming* (SDP) problem is, like *Quadratic Programming*, a convex optimization problem. In an SDP, the feasible set is a cone formed by positive semidefinite matrices and it is expressed as an LMI. Hence, a main advantage is that it has a linear objective function. The SDP form used in this work is the following:

$$\begin{aligned} \min_{\boldsymbol{x}} . \quad & \boldsymbol{\rho}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{F}_0 + \boldsymbol{F}_1 x_1 + ... + \boldsymbol{F}_j x_j \preceq \boldsymbol{0} \\ & \boldsymbol{E}\boldsymbol{x} = \boldsymbol{g} \,, \end{aligned} \tag{67}$$

where $\boldsymbol{\rho} \in \mathbb{R}^j$ is a coefficient column vector, $\boldsymbol{x} \in \mathbb{R}^j$ is the optimization variable and $\boldsymbol{F}_i \in \mathbb{R}^{n \times n}$ are symmetric matrices forming the LMI. j and n are the dimension of the

optimization variable and the symmetric matrix space respectively. The "$\preceq$" symbol stands for negative semi-definiteness. Note that SDP problems can also handle equality constraints.

Our target here is to express the QP problem in (34) as an SDP problem with the form of (67). Overall, we see two main differences between these two problem formulations: the objective functions (quadratic versus linear) and its constraints (linear inequalities versus LMIs). The main actions to move from QP to SDP are:

- **Objective function:** We will express the problem in its *epigraph form*, *i.e.,* upper-bounding the quadratic objective function. This will lead us to an expression that, by means of the *Schur's complement*, can be converted to an LMI. Both epigraph form and Schur's complement are further detailed in Appendices A.2 and A.3, respectively.
- **Constraints:** One can convert a linear inequality to an LMI forming several diagonal matrices with the vector $\boldsymbol{d}$ and every column of $\boldsymbol{C}$ in (34).

Further details on these actions are described hereafter.

Objective Function

To transform the objective function from quadratic to a linear form we use the epigraph form. The underlying concept of this technique is to create an inequality by upper-bounding the objective function with a new variable. This new inequality is introduced as an LMI constraint of the SDP problem and now the optimization problem becomes a minimization of the new variable subject to the created constraint. We refer the reader to Appendix A.2 for further details on the epigraph form.

Following this procedure, we add a scalar upper-bound γ to the cost-function in (34), obtaining

$$\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{l}^\top \boldsymbol{x} \leq \gamma\,. \tag{68}$$

Notice how here u has been also omitted as they are equivalent problems (see Section 2.4.2 for a detailed justification).

Using the *Schur's complement*, (68) can be converted into the following LMI

$$\boldsymbol{M}(\boldsymbol{x}, \gamma) = \begin{bmatrix} \gamma - \boldsymbol{l}^\top \boldsymbol{x} & \boldsymbol{x}^\top \boldsymbol{A}^\top \\ \boldsymbol{A}\boldsymbol{x} & \boldsymbol{I} \end{bmatrix} \succeq 0. \tag{69}$$

Thus, the SDP equivalent to an unbounded QP problem would be

$$\begin{aligned} \min_{\boldsymbol{x},\gamma} . \quad & \gamma \\ \text{s.t.} \quad & -\boldsymbol{M}(\boldsymbol{x},\gamma) \preceq 0 . \end{aligned} \tag{70}$$

This problem can be rewritten so that there is only one optimization variable, namely $\widetilde{\boldsymbol{x}} = [\boldsymbol{x}^\top, \gamma]^\top$. Thus,

$$\begin{aligned} \min_{\widetilde{\boldsymbol{x}}} . \quad & \boldsymbol{\rho}^\top \widetilde{\boldsymbol{x}} \\ \text{s.t.} \quad & -\boldsymbol{M}(\widetilde{\boldsymbol{x}}) \preceq 0 . \end{aligned} \tag{71}$$

Recalling that $\boldsymbol{x} \in \mathbb{R}^j$, we can see in (69) that $\boldsymbol{\rho} \in \mathbb{R}^{j+1}$ is a column vector full of zeros in exception of the last component which is a one. That is the coefficient multiplying the variable γ. In order to gain insight into the dimensions of $\boldsymbol{M}(\widetilde{\boldsymbol{x}})$ we will first look at the dimensions of $\boldsymbol{A}$. $\boldsymbol{A}$ is a linear mapping that maps $\boldsymbol{x} \in \mathbb{R}^j$ towards $\mathbb{R}^r$, thus $\boldsymbol{A} \in \mathbb{R}^{r \times j}$. Knowing this we can deduce that $\boldsymbol{M}(\widetilde{\boldsymbol{x}}) \in \mathbb{R}^{r+1 \times r+1}$. Assimilating $\boldsymbol{M}(\widetilde{\boldsymbol{x}})$ to the LMI $\boldsymbol{F}(\widetilde{\boldsymbol{x}}) \in \mathbb{R}^{n \times n}$ in (67), we see that $n = r + 1$.

Note that in (70), by using the *Schur inequality*, we have introduced a quadratic constraint to the SDP problem. Although in this case this constraint is closely related with the objective function, we could introduce -in the same manner- a quadratic inequality constraint independent of the cost function. Something not possible with QP problems.

We refer the reader to Appendix A.3 for further details on Schur's complement.

Linear Inequality

To deal with the linear inequalities expressed as $\boldsymbol{C}\boldsymbol{x} \preceq \boldsymbol{d}$, if we consider the rows of $\boldsymbol{C} \in \mathbb{R}^{s \times j}$ (being s the number of individual inequalities in $\boldsymbol{C}$) and the elements in $\boldsymbol{d}$ such that

$$\begin{aligned} \boldsymbol{C} &= \begin{bmatrix} \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_s \end{bmatrix} , \\ \boldsymbol{d} &= \begin{bmatrix} d_1, \cdots, d_s \end{bmatrix} , \end{aligned} \tag{72}$$

the inequality can be expressed as the following LMI:

$$\widetilde{\boldsymbol{C}}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{c}_1^\top \boldsymbol{x} - d_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{c}_s^\top \boldsymbol{x} - d_s \end{bmatrix} \preceq 0. \tag{73}$$

Now, our aim is to introduce this LMI in the problem stated at (71). In order to do so, we have to express $\widetilde{\boldsymbol{C}}(\boldsymbol{x})$ in terms of $\widetilde{\boldsymbol{x}}$. This is a simple procedure that consists only in adding a zero column vector at the end of $\boldsymbol{C}$. Doing this, the matrix $\widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})$ would look like

$$\widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}}) = \begin{bmatrix} [\boldsymbol{c}_1^\top, 0]\widetilde{\boldsymbol{x}} - d_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & [\boldsymbol{c}_s^\top, 0]\widetilde{\boldsymbol{x}} - d_s \end{bmatrix} \preceq 0. \tag{74}$$

If we define now two positive semidefinite matrices $\boldsymbol{R} \succeq 0$ and $\boldsymbol{S} \succeq 0$. We can assert that the block-diagonal matrix formed by these two matrices will also be positive semidefinite, *i.e., blkdiag* $(\boldsymbol{R}, \boldsymbol{S}) \succeq 0$. Using this property we can introduce the linear constraint $\boldsymbol{Cx}-\boldsymbol{d}$ into the SDP problem in (71)

$$\begin{aligned} \min_{\widetilde{\boldsymbol{x}}} \quad & \boldsymbol{\rho}^\top \widetilde{\boldsymbol{x}} \\ \text{s.t.} \quad & diag\left(-\boldsymbol{M}(\widetilde{\boldsymbol{x}}), \widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})\right) \preceq 0\,. \end{aligned} \tag{75}$$

In (75), $\boldsymbol{\rho}$ remains unchanged with respect to (71). However, we have now $\boldsymbol{F}(\widetilde{\boldsymbol{x}}) = diag\left(-\boldsymbol{M}(\widetilde{\boldsymbol{x}}), \widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})\right)$, and its dimension has changed to $n = r + s + 1$. Note how the dimension of $\widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})$ has been added. Although this matrix grow considerably, the vast majority of the elements in $\boldsymbol{F}(\widetilde{\boldsymbol{x}})$ is 0. Specialized solvers in SDP problems, take advantage of this sparsity to increase its performance.

With the tools above presented we are able to solve hierarchically any kind of task. In the case of having an equality task it would be exactly the same case as (71). On the other hand, if we have an inequality task we should mount an equivalent problem to (44). With that aim, the optimization variable is augmented adding the slack variable. Then we should introduce the quadratic function minimizing the slack variables analogously to (71). The last step is to add the inequality constraints containing also the slack variables. This can be done using the procedure where we have added the $\widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})$ matrix to the SDP problem. Inequality hard constraints due to higher priority optimizations can also be added as done with $\widetilde{\boldsymbol{C}}(\widetilde{\boldsymbol{x}})$.

3.2 Hierarchical task control through lexicographic SDP

The SDP problem stated above can be tailored to compute robot trajectories to accomplish prioritized tasks (*i.e.,* hierarchy). In this section, as a matter of example, we go through a complete and detailed SDP iteration, specifying carefully all involved matrices. Then, we present the main algorithm to solve the hierarchical task control through the lexicographic SDP approach.

Let us consider a robot with j DoFs. In this example, we want to control two tasks: one inequality and one equality, having the inequality task higher priority than the latter. Both tasks can be expressed in the acceleration domain as

$$\begin{aligned} \boldsymbol{J}_1\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}_1 &\preceq \mathbf{0}\,, \\ \boldsymbol{J}_2\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}_2 &= \mathbf{0}\,. \end{aligned} \tag{76}$$

In the case of a CLIK problem expressed in the acceleration domain, as shown in Section 2.5.2,

$$\widetilde{\ddot{\boldsymbol{\sigma}}}_i = (\ddot{\boldsymbol{\sigma}}_{d,i} - \dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}}_i) + (\lambda_1 + \lambda_2)(\dot{\boldsymbol{\sigma}}_{d,i} - \boldsymbol{J}_i\dot{\boldsymbol{q}}_i) + \lambda_1\lambda_2(\boldsymbol{\sigma}_{d,i} - \boldsymbol{\sigma}_i) \quad i = 1, 2\,. \tag{77}$$

In order to find the values of $\ddot{\boldsymbol{q}}$ that drive our robot joints to accomplish the tasks, we first solve an SDP optimization for the task with higher priority, *i.e.,* the inequality task.

Now, we consider that we have already solved the SDP problem for the first task, being $\ddot{\boldsymbol{q}}^1 \in \mathbb{R}^j$ its solution. For the sake of simplicity, in this example we consider that the inequality task can be accomplished and therefore all slack variables are 0. To solve for the second task, we formulate a second nested SDP problem (lexicographic approach [11] and [12]), defined as

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}},\gamma}\ & .\ \gamma \\ \text{s.t.}\quad & diag(-\boldsymbol{M}(\ddot{\boldsymbol{q}},\gamma), \widetilde{\boldsymbol{C}}(\ddot{\boldsymbol{q}},\gamma) \preceq 0\,, \end{aligned} \tag{78}$$

with $\boldsymbol{M}(\dot{\boldsymbol{q}},\gamma)$ the following LMI:

$$\boldsymbol{M}(\ddot{\boldsymbol{q}},\gamma) = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix} + \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\gamma + \begin{bmatrix} -m_1 & \boldsymbol{j}_{1,2}^\top \\ \boldsymbol{j}_{1,2} & \mathbf{0} \end{bmatrix}\ddot{q}_1 + \ \cdots\ + \begin{bmatrix} -m_j & \boldsymbol{j}_{j,2}^\top \\ \boldsymbol{j}_{j,2} & \mathbf{0} \end{bmatrix}\ddot{q}_j\,, \tag{79}$$

being $\boldsymbol{m}_2^\top = [m_1, ..., m_j] = -2\dot{\boldsymbol{\sigma}}_2^\top\boldsymbol{J}_2$ and $\boldsymbol{j}_{i,2}$ the i-th column of the matrix $\boldsymbol{J}_2$ such that $\boldsymbol{J}_2 = [\boldsymbol{j}_{1,2}, \cdots, \boldsymbol{j}_{j,2}]$.

The LMI $\widetilde{\boldsymbol{C}}(\ddot{\boldsymbol{q}},\gamma) \preceq \mathbf{0}$ contains the information coming from solving the first inequality task and therefore, ensures the hierarchy among the two tasks. In this case,

$$\widetilde{\boldsymbol{C}}(\ddot{\boldsymbol{q}},\gamma) = -diag(\dot{\boldsymbol{\sigma}}_1) + \begin{bmatrix}\mathbf{0}\end{bmatrix}\gamma + diag(\boldsymbol{j}_{1,1})\dot{q}_1 + \ \cdots\ + diag(\boldsymbol{j}_{j,1})\dot{q}_j\,, \tag{80}$$

where $diag(\boldsymbol{a})$ stands for an all-zeros matrix in exception for the elements of its diagonal which are the elements of the vector $\boldsymbol{a}$ vector, *i.e.,* $\dot{\boldsymbol{\sigma}}_1$ and the columns of the matrix $\boldsymbol{J}_1 = [\boldsymbol{j}_{1,1}, \cdots, \boldsymbol{j}_{j,1}]$. Note that $\widetilde{\boldsymbol{C}}(\ddot{\boldsymbol{q}},\gamma)$ will be formed by squared symmetric matrices in $\mathbb{R}^{r\times r}$, where r is the dimension os task 1. Since the slack variables in the first optimization were all zeros, notice also how $\ddot{\boldsymbol{q}}^1$ does not play a role in the second SDP (equality task)(78), similarly as in the QP problem we presented in (45).

Lexicographic SDP algorithm

Algorithm 1 shows how to solve the Lexicographic SDP problem in a practical way. This algorithm has to be run at every time step and goes through all tasks in the hierarchy. The problem we solve is the CLIK in the acceleration domain. Therefore, the input data are the different tasks which are formed by the matrices $\boldsymbol{J}$ and $\widetilde{\ddot{\boldsymbol{\sigma}}}$ as in (76). As an example of this input data, in Section 4, we define a series of tasks devoted to autonomously drive a UAM. The outputs of the algorithm are acceleration references for the robot joints, to drive the robot accomplishing the tasks.

The algorithm is an iteration procedure that goes through all priority levels, starting from the higher one.

We start by storing all task in an input array, namely $\mathcal{T}$. Those tasks with priority equal to the one evaluated at the current iteration, p_{curr}, are taken; and their corresponding matrices $\boldsymbol{J}$ and $\widetilde{\ddot{\boldsymbol{\sigma}}}$ are stored in the arrays $\boldsymbol{E}_{curr}$ and $\boldsymbol{g}_{curr}$ if the evaluated task is equality, or $\boldsymbol{C}_{curr}$ and $\boldsymbol{d}_{curr}$ if it is an inequality.

$\boldsymbol{C}_{curr} \in \mathbb{R}^{r_e \times j}$, $\boldsymbol{g}_{curr} \in \mathbb{R}^{r_e}$, $\boldsymbol{C}_{curr} \in \mathbb{R}^{r_i \times j}$ and $\boldsymbol{d}_{curr} \in \mathbb{R}^{r_i}$, with j the number of robot joints and r_e and r_i are the sum of dimensions of all equality or inequality tasks, respectively, for the current priority level (in Algorithm 1, this procedure is represented by the functions $GetEqualityTasks(\mathcal{T}, p_{curr})$ and $GetInequalityTasks(\mathcal{T}, p_{curr})$).

Once the matrices $\boldsymbol{J}$ and $\widetilde{\ddot{\boldsymbol{\sigma}}}$ of all tasks are loaded, we are ready to mount the SDP problem ($MountObjFuncLMIs(\boldsymbol{C}_{curr}, \boldsymbol{d}_{curr}, \boldsymbol{E}_{curr}, \boldsymbol{g}_{curr})$ and $MountConstraintsLMIs(\boldsymbol{C}_{curr}$, $\boldsymbol{d}_{curr}, \boldsymbol{E}_{curr}, \boldsymbol{g}_{curr})$ in Algorithm 1).

In order to provide priority to inequality tasks as well as to avoid unfeasibility issues (stability), slack variables are used as in Section 2.4.2. Hence, the decision variable will vary in each priority level depending on the shape of $\boldsymbol{C}_{curr}$

$$\boldsymbol{z} = \begin{bmatrix} \ddot{\boldsymbol{q}}^\top & \boldsymbol{w}^\top & \gamma \end{bmatrix}^\top , \tag{81}$$

where $\ddot{\boldsymbol{q}} \in \mathbb{R}^j$ are the accelerations of the joints, $\boldsymbol{w} \in \mathbb{R}^{r_i}$ a vector of slack variables and γ a scalar variable related with the epigraph form presented in (68).

In the Algorithm 1, we have distinguished two kinds of LMIs depending of its origin. On one hand, there are the LMIs $\boldsymbol{M}(\boldsymbol{z})$ related to those constraints coming from tasks with priorities equal to p_{curr}, defined by

$$\boldsymbol{M}(\boldsymbol{z}) = diag(\boldsymbol{M}_1(\boldsymbol{z}), \boldsymbol{M}_2(\boldsymbol{z})) , \tag{82}$$

Algorithm 1: LMI lexicographic optimization algorithm

Input Data: $\mathcal{T} = \{t_1, \ldots, t_\tau\}$ an array of tasks, with τ the total number of tasks. t_i is a task with priority p_i and is expressed in the form $\boldsymbol{J}_i\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}_i \leq \boldsymbol{0}$ in case of inequality task, or in the form $\boldsymbol{J}_i\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}_i = \boldsymbol{0}$ in case of equality.

Result: An acceleration reference $\ddot{\boldsymbol{q}}$ for the robot's joints.

begin
- $p_{curr} \leftarrow GetMaxPriority(\mathcal{T})$
- $\boldsymbol{C} \leftarrow \emptyset$; $\boldsymbol{d} \leftarrow \emptyset$
- $\boldsymbol{E} \leftarrow \emptyset$; $\boldsymbol{g} \leftarrow \emptyset$
- **while** $p_{curr} >= 0$ **do**
  - $\boldsymbol{C}_{curr}, \boldsymbol{d}_{curr} \leftarrow GetEqualityTasks(\mathcal{T}, p_{curr})$
  - $\boldsymbol{E}_{curr}, \boldsymbol{g}_{curr} \leftarrow GetInequalityTasks(\mathcal{T}, p_{curr})$
  - $\boldsymbol{M} \leftarrow MountObjFuncLMIs(\boldsymbol{C}_{curr}, \boldsymbol{d}_{curr}, \boldsymbol{E}_{curr}, \boldsymbol{g}_{curr})$
  - $\widetilde{\boldsymbol{C}} \leftarrow MountConstraintsLMIs(\boldsymbol{C}_{curr}, \boldsymbol{d}_{curr}, \boldsymbol{E}_{curr}, \boldsymbol{g}_{curr})$
  - $\ddot{\boldsymbol{q}} \leftarrow SolveSDPProblem(\boldsymbol{M}, \widetilde{\boldsymbol{C}})$
  - $\boldsymbol{E}, \boldsymbol{g} \leftarrow AddMatrices(\boldsymbol{E}_{curr}, \boldsymbol{E}_{curr} \cdot \ddot{\boldsymbol{q}})$
  - **if** $\boldsymbol{C}_{curr} \cdot \ddot{\boldsymbol{q}} < \boldsymbol{d}_{curr}$ **then**
    - $\boldsymbol{C}, \boldsymbol{d} \leftarrow AddMatrices(\boldsymbol{C}_{curr}, \boldsymbol{d}_{curr})$
  - **else**
    - $\boldsymbol{E}, \boldsymbol{g} \leftarrow AddMatrices(\boldsymbol{C}_{curr}, \boldsymbol{C}_{curr} \cdot \ddot{\boldsymbol{q}})$
  - $p_{curr} \leftarrow p_{curr} - 1$

where $\boldsymbol{M}_1(\boldsymbol{z})$ is an LMI with the form of (69)

$$\boldsymbol{M}_1(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{M}_{1,11}(\boldsymbol{z}) & \boldsymbol{M}_{1,12}^{\top}(\boldsymbol{z}) \\ \boldsymbol{M}_{1,12}(\boldsymbol{z}) & \boldsymbol{M}_{1,22} \end{bmatrix} \succeq 0\,, \tag{83}$$

where

$$\begin{gathered} \boldsymbol{M}_{1,11}(\boldsymbol{z}) = [2\boldsymbol{g}_{curr}^{\top}\boldsymbol{E}_{curr}, \boldsymbol{0}_{r_i\times 1}, 1]\boldsymbol{z}\,, \\ \boldsymbol{M}_{1,12}(\boldsymbol{z}) = diag(\boldsymbol{E}_{curr}, \boldsymbol{I}_{r_i}, 0)\boldsymbol{z}\,, \\ \boldsymbol{M}_{1,22}(\boldsymbol{z}) = \boldsymbol{I}_{r_e+r_i}\,, \end{gathered} \tag{84}$$

with $\boldsymbol{0}_{r_i\times 1}$ an all-zeros row-vector of length r_i. The subscript of the identity matrix indicates its dimension.

$\boldsymbol{M}_2(\boldsymbol{z})$ is an LMI with the same form as (73) also considering the slack variables, such as

$$\boldsymbol{M}_2(\boldsymbol{z}) = [\boldsymbol{C}_{curr}, -\boldsymbol{I}_{r_i}, \boldsymbol{0}_{r_i\times 1}]\,\boldsymbol{z} - diag(\boldsymbol{d}_{curr}) \preceq \boldsymbol{0}. \tag{85}$$

On the other hand, there are the LMIs $\widetilde{\boldsymbol{C}}(\boldsymbol{z})$ related to solutions found at higher priority levels, *i.e.,* previous iterations. These are strict inequalities, having no slack variables, therefore they can be expressed as

$$\widetilde{\boldsymbol{C}}(\boldsymbol{z}) = \left[\boldsymbol{C}, \boldsymbol{0}_{r_i\times(r_i+1)}\right]\boldsymbol{z} - diag(\boldsymbol{d}) \preceq \boldsymbol{0}. \tag{86}$$

where $\boldsymbol{0}_{r_i\times(r_i+1)}$ is an all-zero matrix with r_i rows and r_i+1 columns. Notice matrices $\boldsymbol{E}$ and $\boldsymbol{g}$ contain the constraints for past equality tasks or violated inequality tasks.

Both LMIs, $\boldsymbol{M}$ and $\widetilde{\boldsymbol{C}}(\boldsymbol{z})$, are passed to the SDP solver which returns the resulting vector $\boldsymbol{z}_{opt}$. From the optimizaed variables in $\boldsymbol{z}_{opt}$ we are interested only in the joints accelerations $\ddot{\boldsymbol{q}}$.

The last step before running the procedure for the next priority level is to add current tasks as constraints for future iterations. Equality and inequality tasks will be treated slightly different. In case of equality tasks, $\boldsymbol{E}_{curr}$ matrix will be stored directly in the array for equality constraints $\boldsymbol{E}$. A similar procedure will resume for inequality tasks that are accomplished, $\widetilde{\boldsymbol{A}}_{in}$ matrix will be directly stored in the array for inequality constraints $\boldsymbol{A}_{in}$. However, those inequality constraints that have been violated have will be considered equality constraints in further iterations, thus $\boldsymbol{C}_{curr}$ matrix will be stored in $\boldsymbol{E}$. One thing to point out is that when checking whether the inequality constraints have been violated (from line 12 to line 15), the comparison must be done one row of $\boldsymbol{C}_{curr}$ at a time (represented with the $AddMatrices(\cdot)$ functions in Algorithm 1).

3.3 Summary

This chapter presents the core proposal of the thesis. It contains the proposed approach to generate trajectories (*i.e.,* joint accelerations) for redundant robots considering the accomplishment of a prioritized task stack. This approach is based on a lexicographic semidefinite programming, tailored to deal with a hierarchical task control.

In the first part of the chapter, we have given the tools to transform from QP to SDP optimization problems. This requires two main transformations: the objective function has to be transformed to a linear one; and the linear inequalities have to be expressed as LMIs able to be plugged inside an SDP problem.

The chapter concludes with the detailed description of the procedure which allows us to hierarchically control tasks through lexicographic SDP, supported by the related algorithm shown in Algorithm 1.

4. Application to UAMs

In the previous chapter we explained the lexicographic SDP algorithm. It is a matter of the current chapter to prove its feasibility and performance with a real robot. First, we offer a description of the tasks that will be set for the robot. Then, we describe briefly the robot particularities and present the experiment set. We used the same set either in simulation or real case studies. However, we only include real robot experiments because they better validate the proposed approach, having extra value, and for the sake of conciseness. The chapter concludes with the analysis of these experiments.

4.1 UAM tasks

Usually, the main functionality of a robot can be summarized with few words: "take this object and leave it there" or "go from this point to that other point". However, these few words entail complex issues derived from the robot's surrounding, its geometry, etc. For example, a robot's main function could be "to paint a car". In order to paint a car, we must follow a trajectory at a given velocity so that the paint is applied uniformly. Furthermore, we should ensure that the end-effector of the robot moves perpendicular to the surface being painted. We could also face the situation where the part to be painted is inside the car, so there are some parts of the car's body that have to be avoided. As soon as we start detailing the robot's objective, several tasks arise.

Our main objective in this chapter is to move efficiently the end effector of a UAM, providing the necessary tools to navigate or even precisely move the robot during manipulation phases. If we detail the involved tasks, first we would like to avoid obstacles in order to keep the platform integrity. We also want to avoid joints limits, so the platform does not flies over some bounds (*e.g.,* position, velocity or acceleration) and the arm joints do not reach their limits (usually associated to singularity issues). Then we would move the end effector (position and orientation tracking), while alleviating the effects of the arm inertia by aligning the arm center of gravity (CoG) with the vertical vector of the platform whenever it is possible.

With the previous description there is an implicit priority between tasks, here choosen by common sense (*i.e.,* if we do not keep the platform integrity we cannot use the robot anymore; reaching the joints limits we would not finish the mission; and so on). However, what happens if due to the geometrical limitations of the robot all the stated tasks cannot

be accomplished at the same time? Depending on our objective, or the mission phase, we would like to discard one or another task, *e.g.,* in some cases it might be more interesting to keep position tracking, relaxing the orientation of the end effector, or, on the contrary, orient the end effector loosing precise position tracking (*i.e.,* pointing the tool always to same point). That is why, it is common that this kind of hierarchical trajectory generators -such as the one developed in this thesis- come along with priority planners. Taking into account the current state of the robot, this planner changes the tasks priorities accordingly.

Although this is not the core of the thesis, in order to get better results in the real robot experiments, we developed a simplistic version of a priority planner for our UAM. The criteria used to change priorities is presented after the description of all involved tasks. Note how with the lexicographic SDP problem defined in this work the task priorities can be directly changed without provoking discontinuities in the resulting control refereces. This priority change might become an issue when using analytical LS solutions (*e.g.,* see the time vanishing function required in [2]).

Technically speaking, a task can be defined as a kinematic or dynamic goal. Since this work is about trajectory generation, here we only considered kinematic tasks, and assume that the robot low-level controllers will be able to move the joints to the desired values specified by the trajectory generator.

4.1.1 Obstacle avoidance

A simple approach to avoid an obstacle consists in defining a minimum Euclidean distance d_{min} between the UAM and the obstacle that must be respected. This task can be defined with the following squared form:

$$||\boldsymbol{e}_o||^2 \geq d_{min}^2 \rightarrow \sigma_o = d_{min}^2 - ||\boldsymbol{e}_o||^2 \leq 0\,, \tag{87}$$

being $\boldsymbol{e}_o$ the actual Euclidean distance between the UAM and the center of the obstacle $\boldsymbol{e}_o = \boldsymbol{p}_{uam} - \boldsymbol{p}_{obs}$.

To define the minimum distance between the UAM and the obstacle, we can set a robot footprint (*i.e.,* a sphere around the robot with radius r_{uam} including the whole robot inside it) and an obstacle safety region (*i.e.,* a sphere around the obstacle with radius r_{obs} including the parts of the obstacle that we want to avoid), then $d_{min} = r_{uam} + r_{obs}$. Figure 8 shows schematically these distances.

Now, to express this task in the acceleration domain we have to differentiate $\dot{\sigma}_o = \boldsymbol{J}_o\dot{\boldsymbol{q}}$

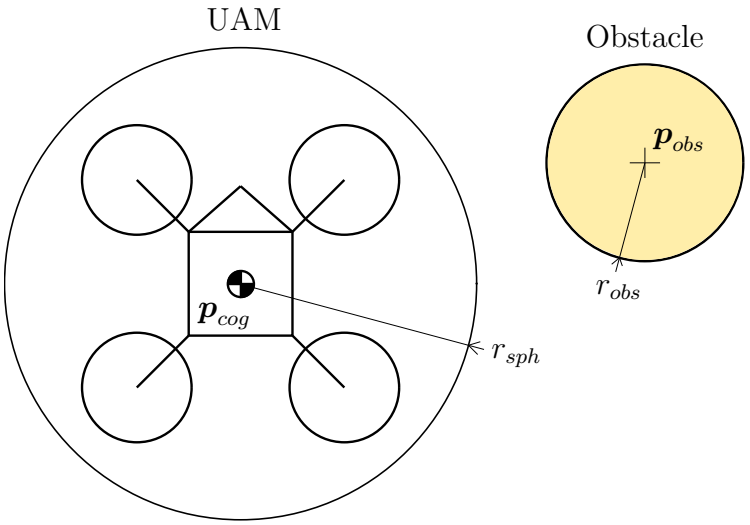


Figure 8: Scheme of distances related to the obstacle avoidance task (top view).

twice with respect to the time and apply the Gronwall's inequality, obtaining

$$\ddot{\sigma}_o = \dot{\boldsymbol{J}}_o\dot{\boldsymbol{q}} + \boldsymbol{J}_o\ddot{\boldsymbol{q}}\,, \tag{88}$$

where $\boldsymbol{J}_o = -2\boldsymbol{e}_o^\top \boldsymbol{J}_{uam}\dot{\boldsymbol{q}}$, which belongs to $\in \mathbb{R}^{1\times j}$, and $\dot{\boldsymbol{J}}_o$ its time derivative. $\boldsymbol{J}_{uam} \in \mathbb{R}^{3\times j}$ is the Jacobian of the UAM body. Then, the task is finally defined by:

$$\widetilde{\ddot{\boldsymbol{\sigma}}}_o = -\dot{\boldsymbol{J}}_o\dot{\boldsymbol{q}} - (\lambda_1+\lambda_2)\boldsymbol{J}_o\dot{\boldsymbol{q}} + \lambda_1\lambda_2\sigma_o\,. \tag{89}$$

Notice how the obstacle avoidance is here defined to avoid collisions with the aerial platform. Hence, we must be aware that the columns of $\boldsymbol{J}_o$ corresponding to the arm joints must be all zeros. Its interpretation is, effectively, that a movement in the arm will not vary σ_o, *i.e.,* it will not move the UAM.

A similar task can be set to avoid self-collisions with the arm, this time involving only terms refered to the arm joints and setting the obstacle radius in the part of the robot that we want to avoid. Its implementation is straight forward and its description is here avoided for the sake of conciseness.

4.1.2 Joint limits

With this task we want to avoid the situation where a joint of our robot reaches its limits. This task is of special interest as those limits can entail some issues such as configuration

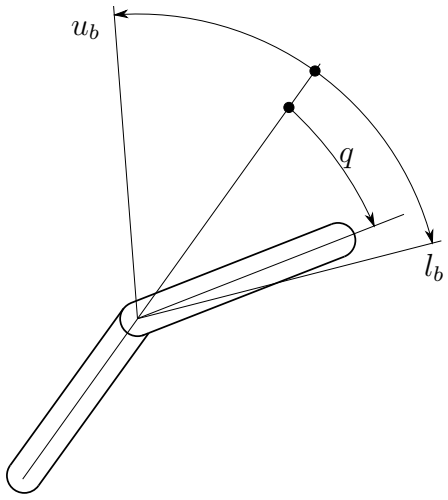


Figure 9: Upper (u_b) and lower (l_b) bounds for a manipulator's joint.

singularities. Due to their geometry, almost all open-link manipulators have joint limits. That is that rather than turn uninterruptedly they are limited by its geometry having upper and lower bounds (see Figure 9). Here we distinguish between two tasks related with position and velocity bounds. Notice that we can also add acceleration bounds directly (*i.e.,* without further transformations) as constraints to the optimization problems as the tasks are defined in the acceleration domain.

Position limits

This task must ensure that values given by the algorithm remain between the joint position bounds,

$$\boldsymbol{l}_b \leq \boldsymbol{q} \leq \boldsymbol{u}_b. \tag{90}$$

being $\boldsymbol{l}_b \in \mathbb{R}^j$ and $\boldsymbol{u}_b \in \mathbb{R}^j$ the vectors with lower and upper bounds for all joints, respectively. Notice that (90) entails two tasks that can be expressed as

$$\begin{aligned} \boldsymbol{\sigma}_{pl,l} &= \boldsymbol{l}_b - \boldsymbol{q} \quad \leq \mathbf{0}\,, \\ \boldsymbol{\sigma}_{pl,u} &= \boldsymbol{q} - \boldsymbol{u}_b \quad \leq \mathbf{0}\,. \end{aligned} \tag{91}$$

Once again, the Gronwall's inequality is used to express this task in the acceleration domain. Only the lower bound inequality will be developed here. The formulation for the upper bound is straight forward with an analogous procedure. Applying two times the Gronwall's inequality we get the following expression:

$$-\ddot{\boldsymbol{q}} - (\boldsymbol{\Lambda}_2 + \boldsymbol{\Lambda}_1)\dot{\boldsymbol{q}} + \boldsymbol{\Lambda}_2\boldsymbol{\Lambda}_1(\boldsymbol{l}_b - \boldsymbol{q}) \leq \mathbf{0}\,, \tag{92}$$

where $\boldsymbol{\Lambda}$ are diagonal gain matrices. Notice that here the task Jacobian has been omitted since for this task $\boldsymbol{J}_{jl,l} = -\boldsymbol{I} \in \mathbb{R}^{j\times j}$. Finally, the task to be introduced in the

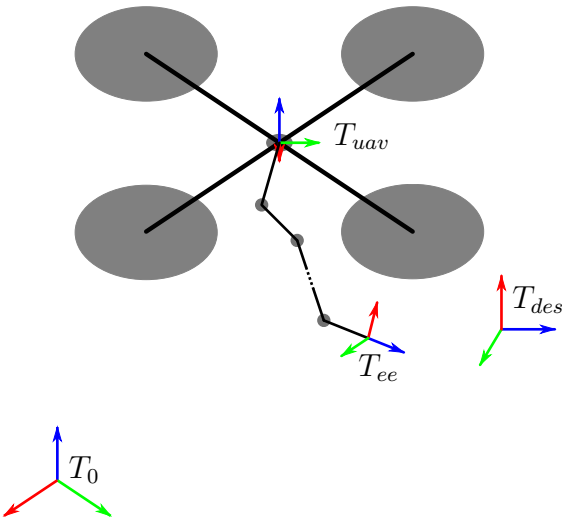


Figure 10: Scheme of UAM frames involved in the pose tracking task (top view).

lexicographic SDP problem is defined as:

$$\widetilde{\ddot{\boldsymbol{\sigma}}}_{pl,l} = (\boldsymbol{\Lambda}_2 + \boldsymbol{\Lambda}_1)\dot{\boldsymbol{q}} - \boldsymbol{\Lambda}_2\boldsymbol{\Lambda}_1(\boldsymbol{l}_b - \boldsymbol{q}) \tag{93}$$

Joint velocity limits

Analogously to the previous task of joint position limits, we can define another task to limit the velocity of the joints with

$$\begin{aligned} \boldsymbol{J}_{vl,l} &= -\boldsymbol{I}\,, \\ \widetilde{\ddot{\boldsymbol{\sigma}}}_{vl,l} &= -\boldsymbol{\Lambda}_1(\boldsymbol{v}_{lb} - \dot{\boldsymbol{q}})\,, \end{aligned} \tag{94}$$

where $\boldsymbol{J}_{vl,l} \in \mathbb{R}^{j\times j}$ is the Jacobian of the task, and $\boldsymbol{v}_{lb} \in \mathbb{R}^j$ is the vector with the lower bounds of the velocities. Similarly to the previous position limits task, we develop here only the task for velocity lower bounds. The formulation for the task considering the velocity upper bounds can be easily derived with an analogous procedure.

4.1.3 Pose tracking

A pose tracking task is the most common task in practically all kind of manipulators as it is intrinsically linked with a manipulator's function: to take objects from one place to an other.

Within this task we consider the position and the orientation of the end effector with respect to a global frame (Figure 10 shows all involved frames for this task). This implies the pose of the platform with respect to the global frame and the robot arm kinematics. The pose of the platform can be provided by any robot state estimation from other robotics fields, *e.g.,* odometry estimation or SLAM. Defining our robot arm through Denavit-Hartenberg parameters, we are able to find the homogeneous transformation between the base of the manipulator (attached to the UAM's body) and the end effector.

Therefore, the end effector pose task can be defined as a column vector such as:

$$\boldsymbol{\sigma}_p^{ee} = \begin{bmatrix} \boldsymbol{p}^{ee} \\ \boldsymbol{\theta}^{ee} \end{bmatrix} , \tag{95}$$

where $\boldsymbol{\sigma}_p^{ee} \in \mathbb{R}^6$ is the end-effector pose, and $\boldsymbol{p}^{ee}(\boldsymbol{q}) \in \mathbb{R}^3$ and $\boldsymbol{\theta}^{ee}(\boldsymbol{q}) \in \mathbb{R}^3$ are the end-effector position and RPY (roll-pitch-yaw Euler angles) orientation, respectively. Notice how subindex p is used to indicate variables related with the pose (not position).

Here, we want to compute the task solving the CLIK problem, thus we must compare the actual and desired values of the task variable, as explained in Section 2.5.2. Hence, the task error and its derivatives are defined as:

$$\boldsymbol{e}_p = \boldsymbol{\sigma}_p^d - \boldsymbol{\sigma}_p \,, \tag{96}$$

$$\dot{\boldsymbol{e}}_p = \dot{\boldsymbol{\sigma}}_p^d - \dot{\boldsymbol{\sigma}}_p = \dot{\boldsymbol{\sigma}}_p^d - \boldsymbol{J}_p\dot{\boldsymbol{q}} \,, \tag{97}$$

$$\ddot{\boldsymbol{e}}_p = \ddot{\boldsymbol{\sigma}}_p^d - \ddot{\boldsymbol{\sigma}}_p = \ddot{\boldsymbol{\sigma}}_p^d - \dot{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \boldsymbol{J}_p\ddot{\boldsymbol{q}} \,. \tag{98}$$

Notice how, for the sake of clarity, we removed the superscript $\bullet^{ee}$ from all variables and the $\bullet^d$ express desired values. The pose Jacobian $\boldsymbol{J}_p \in \mathbb{R}^{6\times j}$ and its time-derivative $\dot{\boldsymbol{J}}_p \in \mathbb{R}^{6\times j}$ can be computed from homogeneous transformations using different methods included in several generalist books like [26].

Now, plugging (96)-(98) into the Gronwall's inequality shown at (57) we have that

$$(\ddot{\boldsymbol{\sigma}}_p^d - \dot{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \boldsymbol{J}_p\ddot{\boldsymbol{q}}) + (\boldsymbol{\Lambda}_1 + \boldsymbol{\Lambda}_2)(\dot{\boldsymbol{\sigma}}_p^d - \boldsymbol{J}_p\dot{\boldsymbol{q}}) + \boldsymbol{\Lambda}_1\boldsymbol{\Lambda}_2(\boldsymbol{\sigma}_p^d - \boldsymbol{\sigma}_p) = 0 \tag{99}$$

where $\boldsymbol{\Lambda}_{1,2}$ are diagonal matrices to assign the desired dynamics to the error. The last thing to do is to assimilate this equation to $\boldsymbol{J}\ddot{\boldsymbol{q}} - \widetilde{\ddot{\boldsymbol{\sigma}}}$ so the task can be solved by means of the lexicographic SDP. Hence,

$$\widetilde{\ddot{\boldsymbol{\sigma}}} = (\ddot{\boldsymbol{\sigma}}_p^d - \dot{\boldsymbol{J}}_p\dot{\boldsymbol{q}}) + (\boldsymbol{\Lambda}_1 + \boldsymbol{\Lambda}_2)(\dot{\boldsymbol{\sigma}}_p^d - \boldsymbol{J}_p\dot{\boldsymbol{q}}) + \boldsymbol{\Lambda}_1\boldsymbol{\Lambda}_2(\boldsymbol{\sigma}_p^d - \boldsymbol{\sigma}_p) \,. \tag{100}$$

Manipulating with a UAM

In a typical manipulator, such as a "grounded" robotic arm, all joints can be used in order to achieve a desired pose since all of them can be actuated with a similar precision. This is not the case of a UAM. Due to physical differences, the joints of the platform can be actuated with less precision than those placed at the arm (the main influences are the bigger mass they move and the nature of their actuators). Thus, it seems reasonable to separate the task of grasping an object (*i.e.,* the pose tracking task) into two tasks.

In order to fulfill this behavior, we have tackled the pose tracking task in the following manner. We define a zone around the manipulation target coincident with the UAM manipulator workspace. Inside that zone, we will perform the end effector tracking only with the manipulator to achieve precise movements, leaving the platform in a hovering state. Outside that manipulation zone, we consider the mission phase as navigation, thus we want to perform large robot displacements only with the robot platform. It would have no sense trying to reach a target with the arm if this is outside the arm workspace.

With this consideration, we can split the *pose tracking* task into the following tasks

- **UAM pose tracking**

$$\boldsymbol{J}_p = \boldsymbol{J}_{uam} \tag{101}$$

$$\widetilde{\ddot{\boldsymbol{\sigma}}}_p = (\ddot{\boldsymbol{\sigma}}_p^d - \dot{\boldsymbol{J}}_p\dot{\boldsymbol{q}}) + (\boldsymbol{\Lambda}_1 + \boldsymbol{\Lambda}_2)(\dot{\boldsymbol{\sigma}}_p^d - \boldsymbol{J}_p\dot{\boldsymbol{q}}) + \boldsymbol{\Lambda}_1\boldsymbol{\Lambda}_2(\boldsymbol{\sigma}_p^d - \boldsymbol{\sigma}_p)\,. \tag{102}$$

  where $\boldsymbol{\sigma}_p \in \mathbb{R}^4$ is the pose of the quadcopter (controlled using 4 DoF as we are not directly controlling neither the roll or the pitch angles of the platform). $\boldsymbol{\sigma}_p^d \in \mathbb{R}^4$ is the desired pose that has been offset from $\boldsymbol{\sigma}_p^d$ the distance of the arm workspace. Therefore, $\boldsymbol{J}_p \in \mathbb{R}^{4\times j}$, and the columns related with the joints in the arm will all be $\mathbf{0}$.

- **End-effector pose tracking**

  The end-effector pose tracking will have the same form as in (100). The only difference will be at the Jacobian $\boldsymbol{J}_p$, where the columns related with the joints of the platform will be all $\mathbf{0}$. That will force the controller to use the arm joints to reach the final end-effector desired pose.

4.1.4 Center of gravity alignment

When carrying a load suspended from an aerial platform, if the center of gravity of this load is not properly aligned with the center of gravity of the body, rotating torques might

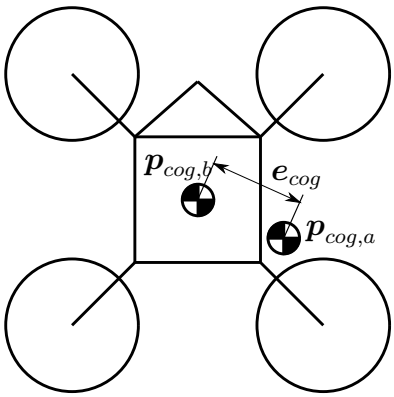


Figure 11: Top view of arm's CoG misalignment

appear in the platform base. This is an undesired situation since these torques have to be compensated by the platform's actuators, situation that leads, among other drawbacks, to an unnecessary consume of energy.

To kinematically avoid this situation during the navigation phase, the joints in the arm work in favor of this alignment. Recall that we only use the platform for large displacements, thus the arm joints can remain free to accomplish secondary tasks. Hence, this task should try to minimize the distance between the arm CoG and the platform vertical vector (*i.e.,* platform gravity vector). This results in reducing the distances between the x and y components of the body CoG, namely $\boldsymbol{p}_{cog,b} \in \mathbb{R}^2$, and the arm's CoG $\boldsymbol{p}_{cog,a} \in \mathbb{R}^2$. Figure 11 shows schematically this situation.

Then, ideally we want

$$\sigma_{cog} = ||\boldsymbol{e}_{cog}||^2 = 0 \tag{103}$$

where $\boldsymbol{e}_{cog} = \boldsymbol{p}_{cog,b} - \boldsymbol{p}_{cog,a}$.

Analogously to the Section 4.1.1, this task is finally defined as:

$$\begin{aligned} \boldsymbol{J}_{cog} &= -2\boldsymbol{e}_{cog}^{\top}\boldsymbol{J}_{cog}\dot{\boldsymbol{q}} \\ \widetilde{\ddot{\boldsymbol{\sigma}}}_{cog} &= -\dot{\boldsymbol{J}}_{cog}\dot{\boldsymbol{q}} - (\lambda_1+\lambda_2)\boldsymbol{J}_{cog}\dot{\boldsymbol{q}} + \lambda_1\lambda_2\sigma_{cog} \end{aligned}, \tag{104}$$

where $\boldsymbol{J}_{cog} \in \mathbb{R}^{2\times j}$ is the Jacobian of the UAM body's center of gravity, considering only the horizontal coordinates.

4.1.5 Acceleration minimization

Once all the tasks described above have been solved we will get joint accelerations to accomplish them. This solution is not necessarily unique, it can belong to a set of joints accelerations that ensure the accomplishment of all tasks.

As we said before, we want to get a smoother behavior in the resulting trajectories. To do so, we have created this acceleration minimization task that is added at the end the tasks stack (*i.e.,* it has the lowest priority). It ensures that, among all the possible accelerations, the minimum norm acceleration will be taken.

This task can be expressed as:

$$\boldsymbol{\sigma}_{acc} = \ddot{\boldsymbol{q}} = \mathbf{0}\,, \tag{105}$$

as it is already in the acceleration domain, its formulation is straightforward with

$$\begin{aligned} \boldsymbol{J}_{acc} &= \boldsymbol{I}_j\,, \\ \widetilde{\ddot{\boldsymbol{\sigma}}}_{acc} &= \mathbf{0}\,. \end{aligned} \tag{106}$$

4.2 Experiment setup

In this section, we describe the setting chosen for the experiments. First, we describe the robot and its particularities. Then, we explain the procedure followed during the real robot experiments. We also remark some notes about the priority planner, *e.g.,* the priorities chosen in respective missions.

4.2.1 Robot description - Kinton UAM

A UAM is a robot composed by an aerial platform, usually a multirotor, and one or several serial manipulators. In this experiments we take advantage of the Kinton robot, both in its simulated and real versions, a UAM from the *Institut de Robòtica i Informàtica Industrial* formed by a quadrotor and a serial arm, shown in Figure 12.

This robot has a scheme of cascade controllers. At the lowest level of control, it has proportional-integral-derivative (PID) controllers for the arm joints, actuated by joint positions, and speed controllers (ESC) for the brushless motors of the platform. On top of these platform ESCs, the quadrotor has its own attitude controller to keep its

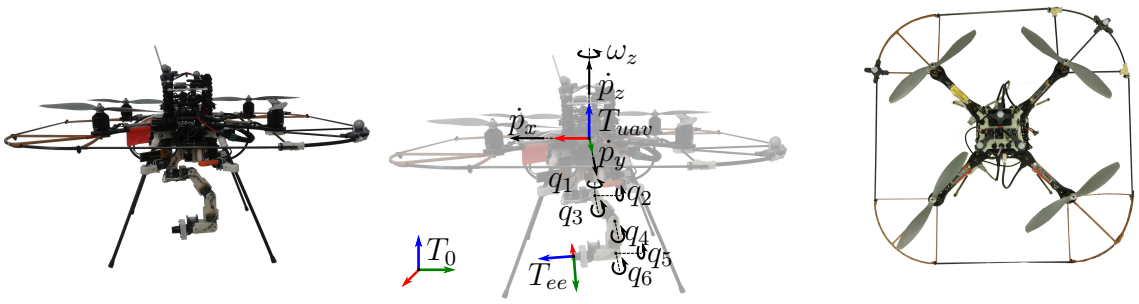


Figure 12: UAM used to perform the experiments - Kinton Robot.

horizontality (*i.e.,* controlling internally the roll and pitch). On top of this attitude controller the quadrotor has a position control scheme described hereafter. Our interest is to send position references to our robot without acrobatic maneuvers, thus we leave the platform roll and pitch as non-controllable DoFs and build the hierarchical task controller on top of the robot positioning system and the arm PIDs. With this, at a high level of control, our robot has a total amount of 10 DoFs, 4 DoFs to control the platform (*i.e.,* position and yaw or heading) and 6 DoFs to control the serial arm.

The flying platform is an ASCTEC Pelican quadrotor which comes with an off-the-shelf attitude controller. By means of the on-board sensors (3-axis accelerometer and 3-axis gyroscope [4]), this controller gives low level commands to the motor drivers (ESCs) to achieve a desired attitude. The drifting phenomena in the on-board sensors difficult the quadrotor position control. Thus, the quadrotor position controller needs extra sensors to localize the platform and close the control loop. In GNSS-denied environments, the position can be obtained with different approaches from other robotics fields such as visual-inertial odometry or SLAM. In this work, however, we are not interested in developing a platform localization method, thus we take advantage of a motion capture system (*i.e.,* Optitrack[5]), already installed at IRI's facilities. This system gives high precision pose measurements, enabling to close the quadrotor position control loop and allowing us to directly send positions to it from our algorithm.

The platform described above is equipped with a 6 DoF robotic arm, which has been specifically designed to be low-weight and to fit the payload of the ASCTEC Pelican quadrotor. The Denavit-Hartenberg parameters that describe the geometry of the arm are shown in Table 1.

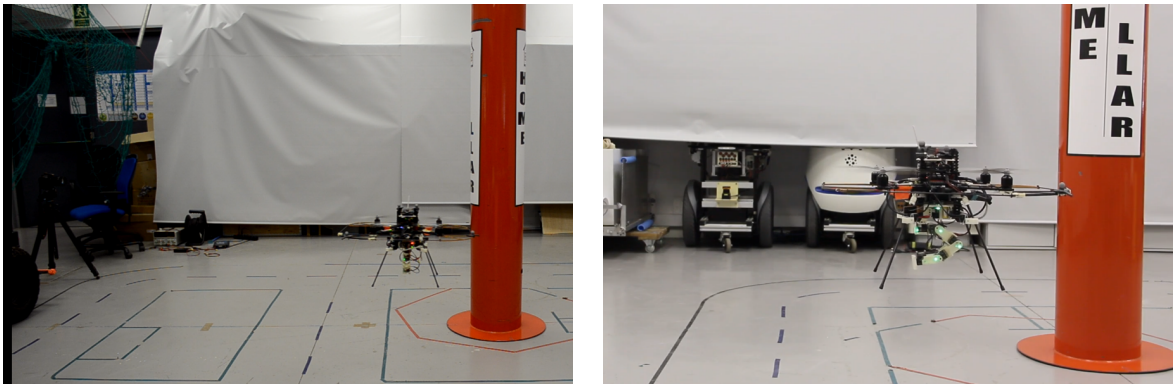
The trajectory generation algorithm of this thesis outputs desired accelerations for the

[4]This kind of sensors are usually delivered within a single microchip that are known as 6 DoF *inertial measurement unit* (IMU).

[5]https://www.optitrack.com

Joint	θ (rad)	d (m)	α (rad)	a (m)
1	q_1	0	0	0
2	$q_2 - \pi/2$	0	$-\pi/2$	0
3	$q_3 - \pi/2$	0	$-\pi/2$	0
4	q_4	0	0	0.065
5	$q_5 + \pi/2$	0.065	0	0.065
6	q_6	0	$\pi/2$	0

Table 1: Denavit-Hartenberg parameters of the 6DoF robot arm used.



(a) Rear view of the UAM during the navigation mode.

(b) Lateral view of the UAM in the manipulation mode.

Figure 13: Samples of real experiments.

joints of the robot that allow to fulfill the desired tasks. These, can be integrated using (63) to obtain desired positions for the arm and quadrotor DoFs, which are the actual inputs of the robot described above. The lexicographic SDP algorithm is programmed with C++ and uses the SDP specialized solver SDPA [27]. All algorithms are running on board, online, using an Intel Atom CPU (@ 1.6GHz) with Ubuntu 16.04LTS and ROS Kinetic.

Figure 13 show two samples of the experiments, during navigation and manipulation modes as described in the following section.

		Tasks					
		Obs. Av.	JL Pos.	JL Vel.	arm CoG alig.	Platform pose	EE pose
Actions	Navigation	1	2	3	4	5	6
	Manipulation	1	2	3	6	5	4

Table 2: Priorities set according to the action being performed by the UAM during the *Task control phase*.

4.2.2 Priority manager

The experiments will consist on a mission where the target is to move the end-effector of the UAM to a specific position and orientation, with the purpose of emulating the navigation and manipulation maneuvers typical of these types of robots. We divide each mission on 3 main phases, all of them carried out autonomously by the UAM. First, the *Take-off phase*. Then, we execute the phase where the task control algorithm runs and moves the UAM to accomplish with the tasks. We call this phase *Task control phase*. We consider that this phase has been successfully accomplished when the norm of the error between the current and desired end-effector poses is smaller than a threshold. Within simulation case studies we set this threshold as $0.05m$ and $0.1rad$. In the real robot experiments we set the values to $0.1m$ and $0.2rad$. In both cases, the position and orientation thresholds are set taking into account the errors in the mechanical design (*i.e.,* the serial arm has been build with rapid prototyping techniques and has some flexibility); the precision of the low-level controllers (*i.e.,* both quadrotor position control and arm PIDs have a small error that cannot be easily reduced); and to finish the missions in a reasonable time once demostrated and proofed the main developments of this thesis. All experiments finish with the *Landing phase*.

From now on, we focus on the important phase, the *Task control phase* where our algorithm runs. During that phase we might want to change the task stack priorities, which is done by the priority manager. The manager is also capable to disable a task if required.

In the manager, we distinguish two priorizations that depend on the distance between the actual and desired end-effector poses: *navigation* and *manipulation*. This distance threshold to switch between priorizations is related with the arm workspace, as mentioned previously in this document, we want to perform large displacements with the platform and precise and small movements with the arm. Considering the tasks detailed in Section 4.1, Table 2 shows the priorities set by the manager.

Notice how we always prioritize the platform integrity and then to respect the joint limits.

During the navigation phase it is important to maintain the arm CoG aligned with the vertical axis of the platform, so the quadrotor motors do not have to counteract torques created by the arm. This means that the arm joints are "locked" by the CoG task during the navigation phase. When the robot is in its *manipulation* mode, however, we release the arm by setting the arm CoG alignment task at the lowest priority.

4.3 Experiments results

In this section, we present the results of real robot experiments to validate the proposed approach. Simulation case studies are here avoided for the sake of conciseness.

To demonstrate the viability of the lexicographic SDP algorithm, we present here the results of a single mission, analizing the effect of each task. Although there exist other hierarchical task control algorithms in the literature, we believe that we cannot directly compare performances with them. By setting equality and also inequality tasks (most of the current methods can only solve equality cost functions); expressing the system through LMIs; using dedicated solvers to this SDP problems (different solvers can produce different results, difficulting the comparison); and with the facilities of our method to exchange priorities between tasks (other methods may require time-vanishing functions to switch priorities); we believe a comparison with others methods could not be fair.

In all experiments, we have set acceleration bounds for the joints as constraints in the optimization processes. In all the following plots we remark with a black dashed vertical line the point in time where the priority manager changes from navigation to manipulation modes, thus producing a change in the task stack by modifying the priorities according to Table 2.

Obstacle avoidance

To clearly see how the obstacle avoidance works, we show in Figure 14 the trajectory done by the UAM during the navigation phase with an obstacle in the middle of the direct trajectory. Without the presence of the obstacle the robot trajectory would result in a direct line between the starting point and the target. However, when an obstacle appears (defined by the red circle) the trajectory is modified to avoid it (red continuous line), mostly satisfying the inflation radius constraint (yellow area plotted, in this case, around the obstacle). In Figure 14 we show a particular result where the inflation radius is violated some centimiters, this case happens when the acceleration bounds set to the

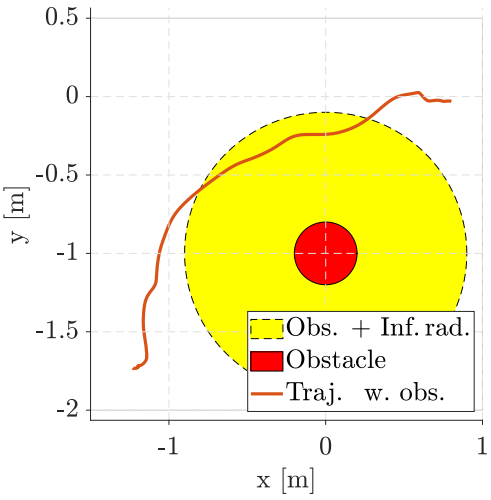


Figure 14: Obstacle avoidance task within a real robot experiment.

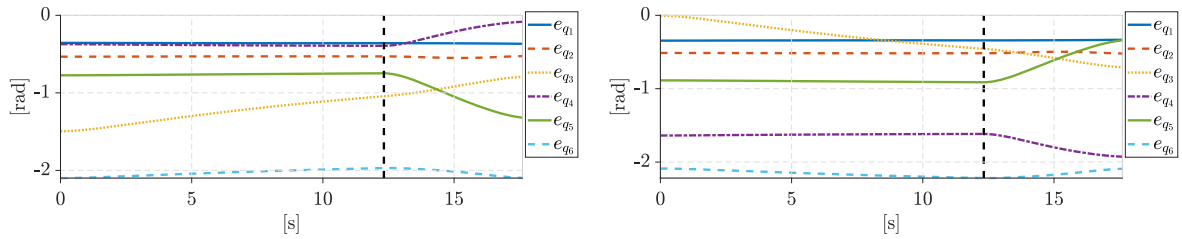
problem are not sufficient to let the optimization produce references to make the robot react in time. Although in this case the obstacle is avoided without any collision, one has to consider this when designing the mission. The obstacle avoidance is also evident in the first part of the Figure 18a, where the motion of the platform is prevented for the x axis (continuous blue line). Once the obstacle has been avoided, the trajectory is resumed (*i.e.,* the blue line is exponentially decreased).

Joint limits - Position

In order to see how the joints are kept within their limits we compute the distance to their respective bounds. Here we show the task set to respect the position bounds. Specifically, Figure 15a shows the joint distances to the upper bounds, defined with $\boldsymbol{q} - \boldsymbol{u}_b$. Similarly, Figure 15b depicts the error between the joints current values and their respective lower-bounds, *i.e.,* $\boldsymbol{l}_b - \boldsymbol{q}$. As any error of both figures is above 0, we can tell that the position arm joint limits are respected.

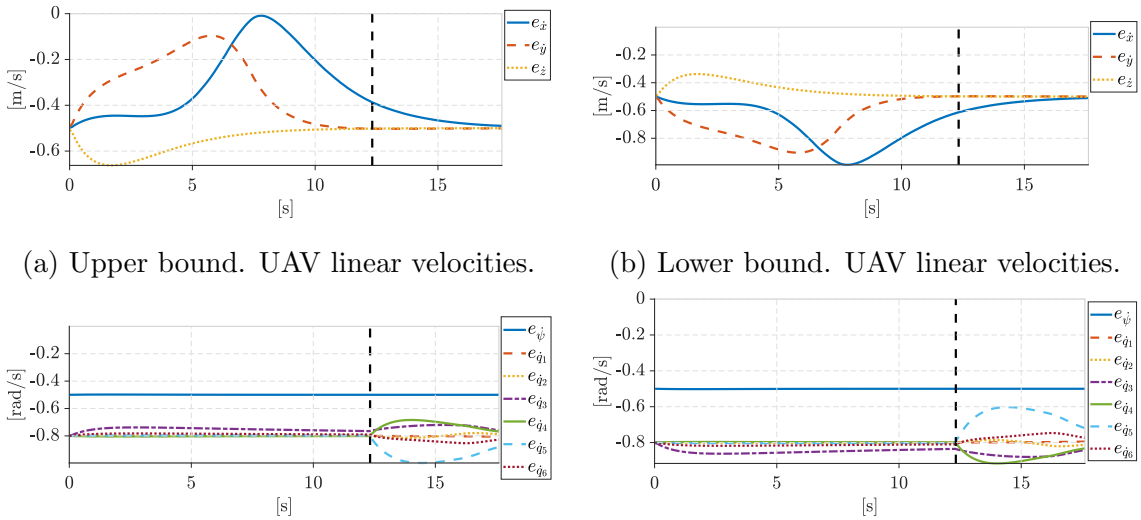
Joint limits - Velocity

The same interpretation as for the task limiting the joints positions can be done for the velocity joint limits task. In this case we differentiate between the translational and rotational joints, that is, Figures 16a and 16b are the plots related to the upper and lower



(a) Upper bound (b) Lower bound

Figure 15: Error between current joints positions and their respective bounds during a real experiment.



(a) Upper bound. UAV linear velocities. (b) Lower bound. UAV linear velocities.

(c) Upper bound. Yaw and arm joints. (d) Lower bound. Yaw and arm joints.

Figure 16: Error between current joints velocities and their respective bounds during a real experiment.

bounds of the linear velocities of the quadrotor, respectively. Figures 16c and 16d are related with the angular velocities (quadrotor yaw and arm joints). Once again, all values are below 0, thus all velocity joint limits are respected.

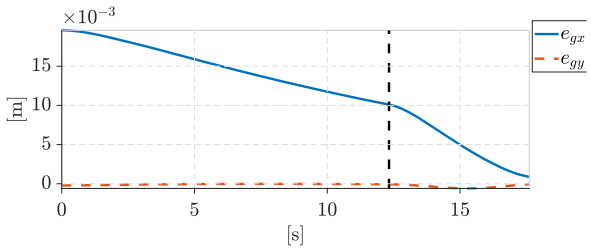


Figure 17: Arm's CoG alignment task during a real robot experiment.

Arm's CoG alignment

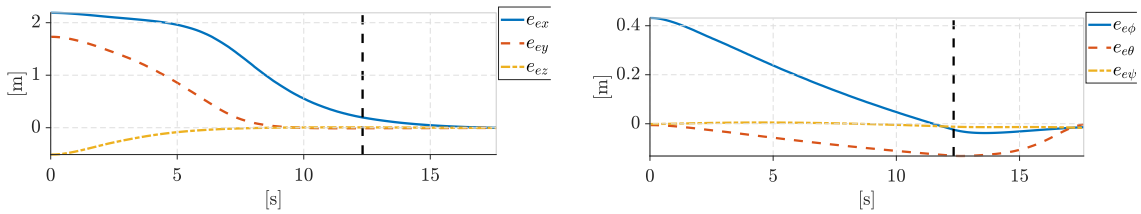
The serial arm has been designed to have the workspace in front of the quadrotor. Altough it has 6 DoFs, its main movements usually include forward-backward (*i.e.,* platform x axis) and up-down (*i.e.,* platform z axis) maneuvers because it has to avoid collisions with the platform legs (the arm operates between the platform frontal legs). This behaviour can be seen in Figure 17 where we show the planar 2D distance error between the arm CoG and the gravitational vector of the platform (x and y dimensions). Although this distance is reduced in both x and y axis of the platform frame, the reduction is heavier in the x direction as the y dimension is already aligned by design.

End-effector pose tracking

The results of applying the end-effector tracking task are shown in Figure 18 by means of the error between the current and desired end-effector poses (translation and orientation). Once again, we show the navigation (left) and manipulation (right) phases separated by a dashed line.

Figure 18a clearly shows that the positioning error $||e_{pos}||$ descends towards $\mathbf{0}$ either in the navigation phase or the grasping phase. Notice how, the error decrease is delayed for the x axis and, in minor measure, for y axis during the initial part of the navigation phase. This behaviour occurs because the aerial vehicle is avoiding an obstacle (see Figure 14) as its task has higher priority. When the obstacle is avoided, the end-effector pose tracking task resumes as expected. As previously mentioned, the task control phase of the mission finishes when the end-effector reaches a certain error with respect to its desired pose, *i.e.,* $||e_{pos}|| < 0.1m$ and $||e_{ori}|| < 0.2rad$.

Figure 18b shows the error related with the end-effector attitude, *i.e.,* $||e_{ori}||$. Notice that we deactivate on purpose the use of the serial arm during navigation phases, in



(a) Position (b) Attitude

Figure 18: End-effector pose tracking error during a real robot experiment.

order to leave the large displacements to the aerial platform and the precise movements to the serial arm during manipulation phases. This behaviour can be identified as the end-effector orientation errors are not decreased towards **0** during the navigation phase (left part of 18b) because the platform has not enough DoFs to do it. Instead, the full 6 DoF error of the end-effector is reduced in the manipulation phase as the serial arm plays a role here with its 6 DoFs.

4.4 Summary

In this chapter, we described a particular real robot application of the method presented in Chapter 3.

First, we defined the tasks implemented, specially designed considering the type of robot, *i.e.,* a UAM. More precisely, the tasks include obstacle avoidance with the highest importance to keep the robot integrity. Then, we avoid the limitis of the robot joints to discard joint issues related to singularities. We improve the flight behaviour by aligning the arm CoG with the gravitational vector of the platform, to reduce the torques produced in the base by its misalignment. Finally, we set our main task to move the arm end-effector towards a desired pose target.

The first section is followed by an explanation of the experiment set, *i.e.,* the facilities, equipment used and the procedure followed to proceed with the experiments. Finally, the chapter ends by showing results of a real robot experiment with plots showing the viability of the proposed approach and the tasks definitions.

Concluding Remarks

Conclusions

Nowadays, complex robots with high number of DoFs are a reality. Their working spaces have high dimensionalities, making complex the control of the joints to perform desired tasks, *i.e.,* to create joint trajectories fulfilling them. In that sense, in this work we tackle this problem by analizing the trajectory generation peculiarities and presenting a task control law. Although the main contributions can be generically applied to redundant robots, we include a particular application to unmanned aerial manipulators to demonstrate its viability.

This work starts with an accurate review of the existing literature on trajectory generation algorithms. In order to place the reader to the specific problem, we describe the formulation of the most relevant methods in detail, giving a solid background and introducing the main contribution of this thesis: a trajectory generation algorithm based on semi-definite programing. In this background chapter, we start describing the well-known least squares formulation, we move to a quadratic problem description and finally to semi-definite programming, with a clear progression for the sake of readibility and comprehension.

The presented approach, in contrast to most existing methods, is able to handle not only tasks defined by an equality cost function but also for inequality tasks. It allows to set a strict hierarchy among all tasks defined in the task stack and to define a task priority manager, able to directly exchange priorities between tasks online. In the event that these tasks are expressed in position or velocity domains, we have provided a procedure to express them and solve the optimal problem in the acceleration domain, thus producing smooth joint trajectories in position domain. The optimal problem formulation in the main core of the thesis is based on linear matrix inequalities, allowing an easy-to-understand formulation and the use of fast and dedicated solvers. All presented formulation developments are accurately explained step by step with simple examples and proofs, with extended explanations in the appendix sections, including a full example in Matlab to play with the different concepts.

In the last part of this thesis, we have demonstrated the viability of the proposed approach with a particular application to unmanned aerial manipulators, describing their particularities and specific tasks. We show the method performance and analize the effect of the

different blocks (mission phases, tasks, priorities, etc.) through real robot experiments.

All code produced within this work will be made available through the IRI gitlab code repository for the benefit of the community.

Future Work

In this thesis we have reached the main proposed objectives, however, during its development we discovered new research paths that we think they are interesting to investigate.

Altough the presented algorithm solves for the tasks successfully, we have been able to corroborate some facts regarding its performance during the experiments. The robot design is highly correlated with the task definitions. When designing the tasks, we have to consider if the robot DoFs will be available to move the joints and solve them. In other words, if we set tasks requiring DoFs not available in the robot, they will not be fulfilled. In this work we carefully designed tasks that can be accomplished within our particular application, but it would be interesting to define a procedure to study the task spaces in advance, to determine if our robot will have the correct DoFs.

Furthermore, it is highly important how a task is formulated so the robot behaves as we expect. For example, the end-effector orientation within the end-effector tracking task is now computed as a difference of angles. We believe that a better way to express this error is by computing it through Lie Algebra in the *Special Orthogonal Group* $SO3$. We considered this development out of the thesis scope and it is here avoided for the sake of simplicity.

Besides, regarding the performance of the algorithm, it would be interesting to implement changes that could reduce the computation time of a single solver iteration. Works such as [15] could be of high interest for that aim.

Finally, in this work we focused on the control law and we implemented a simple task priority manager in the particular application, hence a smarter version of the priority planner can be designed.

References

[1] A. Santamaria-Navarro, P. Grosch, V. Lippiello, J. Sola, and J. Andrade-Cetto, "Uncalibrated Visual Servo for Unmanned Aerial Manipulation," *IEEE/ASME Transactions on Mechatronics*, vol. 22, pp. 1610–1621, Aug. 2017.

[2] V. Lippiello, J. Cacace, A. Santamaria-Navarro, J. Andrade-Cetto, M. A. Trujillo, Y. R. Esteves, and A. Viguria, "Hybrid Visual Servoing With Hierarchical Task Composition for Aerial Manipulation," *IEEE Robotics and Automation Letters*, vol. 1, pp. 259–266, Jan. 2016.

[3] D. E. Whitney, "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 94, pp. 303–309, Dec. 1972.

[4] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-Priority Based Redundancy Control of Robot Manipulators," *The International Journal of Robotics Research*, vol. 6, pp. 3–15, June 1987.

[5] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1990.

[6] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Science & Business Media, Aug. 2010.

[7] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[8] A. Liegeois, "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, pp. 868–871, Dec. 1977.

[9] O. Kanoun, F. Lamiraux, and P. B. Wieber, "Kinematic Control of Redundant Manipulators: Generalizing the Task-Priority Framework to Inequality Task," *IEEE Transactions on Robotics*, vol. 27, pp. 785–792, Aug. 2011.

[10] E. Lutscher and G. Cheng, "Hierarchical inequality task specification for indirect force controlled robots using quadratic programming," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Chicago, IL, USA), pp. 4722–4727, Sept. 2014.

[11] C. Ocampo-Martinez, A. Ingimundarson, V. Puig, and J. Quevedo, "Objective Prioritization Using Lexicographic Minimizers for MPC of Sewer Networks," *IEEE Transactions on Control Systems Technology*, vol. 16, pp. 113–121, Jan. 2008.

[12] S. Siniscalchi-Minna, F. D. Bianchi, and C. Ocampo-Martinez, "Predictive control of wind farms based on lexicographic minimizers for power reserve maximization," in *2018 Annual American Control Conference (ACC)*, pp. 701–706, June 2018.

[13] A. Escande, N. Mansard, and P. B. Wieber, "Fast resolution of hierarchized inverse kinematics with inequality constraints," in *2010 IEEE International Conference on Robotics and Automation*, (Anchorage, AK, USA), pp. 3733–3738, May 2010.

[14] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, pp. 327–363, Dec. 2014.

[15] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, pp. 1006–1028, June 2014.

[16] R. Rossi, A. Santamaria-Navarro, J. Andrade-Cetto, and P. Rocco, "Trajectory Generation for Unmanned Aerial Manipulators Through Quadratic Programming," *IEEE Robotics and Automation Letters*, vol. 2, pp. 389–396, Apr. 2017.

[17] A. De Luca, G. Oriolo, and B. Siciliano, "Robot redundancy resolution at the acceleration level," *Laboratory Robotics and Automation*, vol. 4, pp. 97–106, Jan. 1992.

[18] A. Santamaria-Navarro, V. Lippiello, and J. Andrade-Cetto, "Task priority control for aerial manipulation," in *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, (Hokkaido, Japan), pp. 1–6, Oct. 2014.

[19] F. Caccavale, S. Chiaverini, and B. Siciliano, "Second-order kinematic control of robot manipulators with Jacobian damped least-squares inverse: theory and experiments," *IEEE/ASME Transactions on Mechatronics*, vol. 2, pp. 188–194, Sept. 1997.

[20] G. Antonelli, "Stability Analysis for Prioritized Closed-Loop Inverse Kinematic Algorithms for Redundant Robotic Systems," *IEEE Transactions on Robotics*, vol. 25, pp. 985–994, Oct. 2009.

[21] P. Falco and C. Natale, "On the Stability of Closed-Loop Inverse Kinematics Algorithms for Redundant Robots," *IEEE Transactions on Robotics*, vol. 27, pp. 780–784, Aug. 2011.

[22] F. Flacco and A. De Luca, "Discrete-time redundancy resolution at the velocity level with acceleration/torque optimization properties," *Robotics and Autonomous Systems*, vol. 70, pp. 191–201, Aug. 2015.

[23] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1989.

[24] B. Siciliano and J. J. E. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *, Fifth International Conference on Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR*, (Pisa, Italy), pp. 1211–1216 vol.2, June 1991.

[25] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," *IEEE Transactions on Automatic Control*, vol. 44, pp. 648–654, Mar. 1999.

[26] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*, vol. 3. Wiley New York, 2006.

[27] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto, *A high-performance software package for semidefinite programs: SDPA 7*. 2010.

[28] J. Marti-Saumell, "SDPTrajectoryGeneration: MATLAB code to hierarchically generate trajectories using semidefinite programming.," Aug. 2018. original-date: 2018-07-16T10:51:34Z.

[29] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition*. Springer Tracts in Advanced Robotics, Springer International Publishing, 2 ed., 2017.

A. Theoretical notes

In the development of the thesis formulation there are theoretical concepts that are not directly related with the field of trajectory generation for redundant robots. However, we include the most important ones in this section for the sake of completeness.

The notation used in the sections of this appendix is not directly the one used in the thesis development, but it is a simplified notation to ease the reading.

A.1 Slack Variables

Typically, a slack variable is used in optimization problems to convert from an inequality to an equality constraint. More precisely, it is used in *linear programming* (LP) problems.

The standard LP is a minimization of a problem with the following form:

$$\begin{aligned} \min_{\boldsymbol{x}} . \quad & \boldsymbol{f}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{C}\boldsymbol{x} = \boldsymbol{d} \\ & \boldsymbol{x} \succeq \boldsymbol{0} \,, \end{aligned} \tag{107}$$

where the cost function is a linear expression with $\boldsymbol{f} \in \mathbb{R}^n$ a coefficient column vector multiplying the decision variable $\boldsymbol{x} \in \mathbb{R}^n$, being n the dimension of our ambient space. The LP problem should be solved considering the equality constraints expressed as $\boldsymbol{C}\boldsymbol{x} = \boldsymbol{d}$, with $\boldsymbol{C} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{d} \in \mathbb{R}^m$. The number of columns in the matrix $\boldsymbol{C}$, represented by the variable m, indicates the number of equality constraints present in the LP problem. Therefore, a single equality constraint placed at the i-th row of $\boldsymbol{C}$, can be expressed as $\boldsymbol{c}_i\boldsymbol{x} = d_i$.

According to (107), in the LP standard-form all constraints must be expressed as equality constraints and all variables must be positively defined. However, there could be the case where we want to add an inequality constraint such as $\boldsymbol{a}\boldsymbol{x} - b \leq 0$. In order to express this as an equality constraint we must introduce a slack variable w and modify the constraint as:

$$\boldsymbol{a}\boldsymbol{x} - b + w = 0 \,, \tag{108}$$

with $w \geq 0$. In the case of the inequality constraint (108) is active ($\boldsymbol{a}\boldsymbol{x} - b = 0$), the slack variable w will be 0. On the other hand, if the constraint is inactive, the slack

variable w will take the needed value in order to make (108) hold. The resulting LP problem including the inequality constraint expressed as an equality constraint would be as follows:

$$\begin{aligned} \min_{\widetilde{\boldsymbol{x}}} \; & \boldsymbol{f}^\top \widetilde{\boldsymbol{x}} \\ \text{s.t.} \; & \widetilde{\boldsymbol{C}}\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{d}} \\ & \widetilde{\boldsymbol{x}} \succeq \boldsymbol{0} \,. \end{aligned} \tag{109}$$

Being $\widetilde{\boldsymbol{x}} = [\boldsymbol{x}, w]^\top$ the augmented decision variable including the slack variable w. Similarly, $\widetilde{\boldsymbol{C}}$ and $\widetilde{\boldsymbol{d}}$ have been augmented as follows:

$$\widetilde{\boldsymbol{C}} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{a} & 1 \end{bmatrix} \,, \quad \widetilde{\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{d} \\ b \end{bmatrix} . \tag{110}$$

In the event that we would like to introduce more than one inequality constraint we would follow a similar process to the one recently described. First, we could express them with the following matrix-form expression: $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \preceq \boldsymbol{0}$. In order to include all the inequality constraints in the standard LP problem, we should consider one slack variable for each inequality, *i.e.,* a vector of slack variables $\boldsymbol{w}$ with the same length as $\boldsymbol{b}$. Finally, in (110), the $\boldsymbol{a}$ in $\widetilde{\boldsymbol{C}}$ would be the matrix $\boldsymbol{A}$, the 1 would be an identity matrix and the b in $\widetilde{\boldsymbol{d}}$ would be the vector $\boldsymbol{b}$.

When slack variables are introduced in a trajectory generation algorithm in order to deal with inequality tasks (see section 2.4.2), we are not trying to express an inequality constraint as an equality one. Instead, we are guaranteeing a solution even though some constraints can not be fulfilled.

In the following, we will try to use a simple 1-variable numerical example to show the importance of adding slack variables into the algorithm.

Let us assume that we have 4 tasks with 2 hierarchy levels (2 priorities). For example the following ones:

$$\begin{aligned} x - 3 \leq 0 \quad & (\textit{priority } 1) \,, \\ 0.5x - 2 = 0 \quad & (\textit{priority } 2) \,, \\ -x + 5 \leq 0 \quad & (\textit{priority } 2) \,, \\ x - 3.5 \leq 0 \quad & (\textit{priority } 2) \,. \end{aligned} \tag{111}$$

Solving for the first priority is straightforward. Since it is an inequality task, the optimization problem ends-up being a feasibility problem resulting with a feasible set $x \leq 3$ (shown in blue at Figure 19).

Then, we want to solve for 3 tasks with priority 2: 1 equality task and 2 inequalities. Inequality constraints for the second priority are represented in Figure 20. We can see

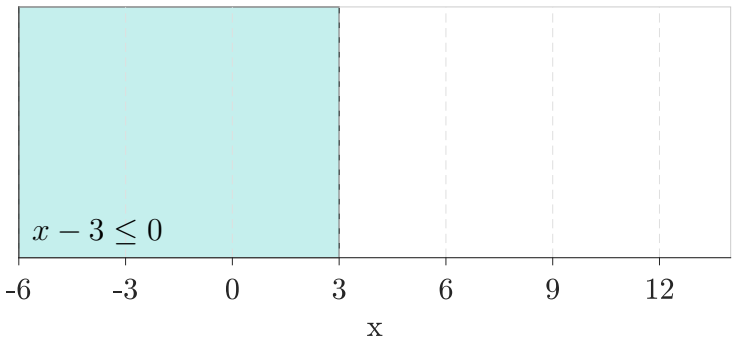


Figure 19: Feasibility region for the first task described in (111).

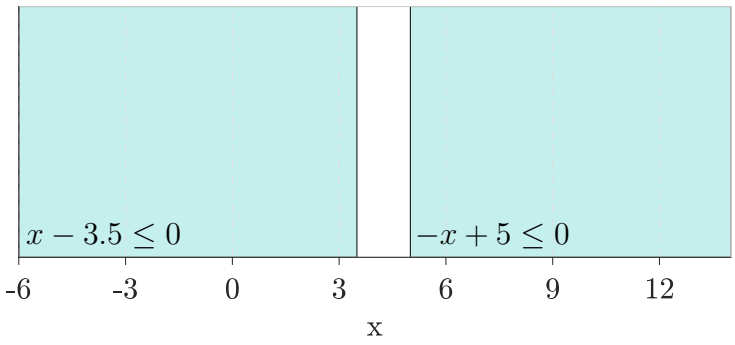


Figure 20: Feasibility region for inequality tasks in the second priority level.

that there is no region where the two tasks are fulfilled. This fact could lead to an unfeasibility issues unless we consider slack variables. Besides, we also have to consider the task with priority 1 as a hard constraint.

Having said that, we may state a QP problem considering two slack variables, each one related with the respective constraints shown at Figure 20: w_1 and w_2. Therefore, the decision variable should be augmented accordingly: $\boldsymbol{x} = [x, w_1, w_2]^\top$. For the sake of clarity, we want now to state the cost function, which should minimize the slack variables as well as the equality task. Both requirements can be expressed as the following quadratic function:

$$\min_{x,w_1,w_2} . \quad ||0.5x - 2||_2^2 + ||\boldsymbol{w}||_2^2 . \tag{112}$$

Now, we are in disposition of stating a final QP problem with $\boldsymbol{x}$ as the decision variable. In order to do so we may expand the cost function in (112). We should also consider the two constraints with priority 2 and the hard constraint with priority 1 as follows:

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} . & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^\top \boldsymbol{x} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \preceq \boldsymbol{0} \end{array} , \tag{113}$$

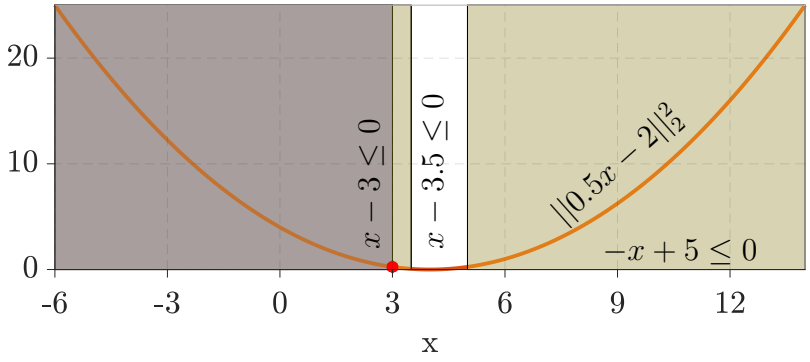


Figure 21: Representation for the tasks in (111). Optimal x marked with a red dot.

where

$$\begin{aligned} \boldsymbol{H} &= \begin{bmatrix} 2 \cdot 0.5^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ \boldsymbol{f}^\top &= [2 \cdot 0.5 \cdot 2, 0, 0], \\ \boldsymbol{A} &= \begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}, \\ \boldsymbol{b}^\top &= [-3, 5, -3.5]. \end{aligned} \tag{114}$$

Notice that in the matrix $\boldsymbol{A}$ the first row, related with the inequality task with priority 1, the columns related with the slack variables are 0 (it is a hard constraint). Now, the cost function aims at minimizing the equality task and the slack variables.

A graphical representation of (113) can be seen in Figure 21. Constraints are indicated with shaded colors and their corresponding equations. The convex curve is the quadratic function related to the equality task $||0.5x - 2 = 0||_2^2$. Recall that constraints of priority 2 can be violated with the drawback of increasing the cost-function value.

The solution for the problem in (113) is $\boldsymbol{x}^* = [3, 2, 0]^\top$ (marked with a red dot in Figure 21). The solution must fulfill the constraint $x - 3 \leq 0$ because it is a hard constraint. We know that the constraint $(-x + 5 \leq 0)$ will be violated (since its intersection with the hard constraint $x - 3 \leq 0$ is a void set), *i.e.,* $w_1 > 0$. Thus, the optimal solution will be a compromise between minimizing the equality task and the slack variable w_1. In the case of the tasks defined in (111), x will be placed at the rightest possible position, *i.e.,* , $x = 3$.

In the given example with tasks defined in (111), the best solution for the second priority is reached keeping $w_1 = 0$. However, it could happen that the cost function minimum is

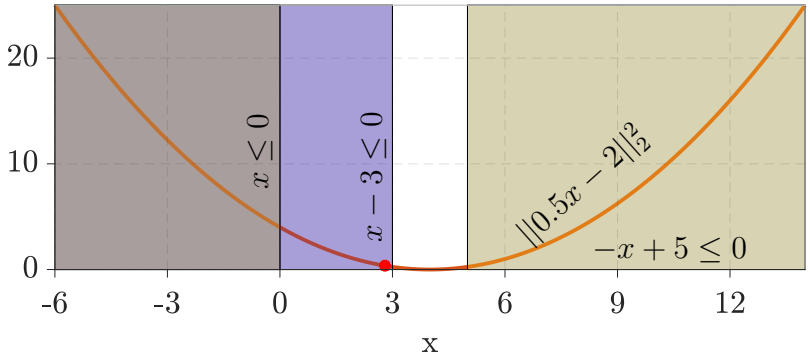


Figure 22: Representation of the QP problem when the inequality task $x - 3.5 \leq 0$ has been replaced by $x \leq 0$. Optimal x marked with a red dot.

reached violating both inequality tasks and, therefore, making $w_1 > 0$ and $w_2 > 0$. This situation can be seen considering the constraint $x \leq 0$ instead of $x - 3.5 \leq 0$ (see Figure 22). Notice that the stated change modifies the optimal solution to $\boldsymbol{x}^* = [2.8, 2.2, 2.8]^\top$ where both inequality constraints are violated.

A.2 Epigraph form

The *epigraph* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the set of points lying above its graph.

Mathematically, this set is expressed as

$$epi(f) = \{(x, \gamma) | x \in \mathbf{dom} f, f(x) < \gamma\} \subseteq \mathbb{R}^{n+1}, \tag{115}$$

where $\mathbf{dom} f$ states for *domain of the function* f. Notice how the epigraph is a set with one dimension more than the dimension of the variable x. This is formed by the values of $x \in \mathbb{R}^n$ belonging to the domain of f adding an extra dimension with those values of a variable γ that are bigger than $f(x)$.

Figure 23 shows a graphical representation of the epigraph of a sample function. A function is said to be convex if its epigraph is a convex set.

Some of the typical optimization problems assume linear objective functions. We can use the epigraph definition to express any optimization problem as a problem with a linear

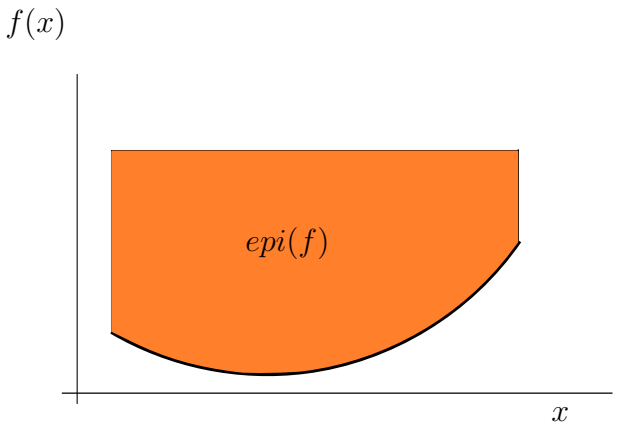


Figure 23: Epigraph of a sample function $f(x)$, shown as the shaded orange area lower-bounded by $f(x)$.

objective function. Therefore, $min.f(x)$ is equivalent to

$$\begin{aligned} \min_{x,\gamma} \; & \gamma \\ \text{s.t.} \; & f(x) - \gamma \leq 0 \end{aligned} . \tag{116}$$

where x and γ are now the optimization variables. This problem formulation is known as the *epigraph form*.

A.3 Schur Complement

The Schur complement is a useful techinique to convert nonlinear constraints into LMI constraints. Let us assume we have a matrix $\boldsymbol{X}$ that can be decomposed into 4 matrices such as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{C} \end{bmatrix} , \tag{117}$$

being $\boldsymbol{A}$ invertible. Then, the matrix $\boldsymbol{S} \triangleq \boldsymbol{C} - \boldsymbol{B}^\top \boldsymbol{A}^{-1} \boldsymbol{B}$ is the Schur complement of X and has the following property

- If $\boldsymbol{A} \succ 0$, then $\boldsymbol{X} \succeq 0$ if and only if $\boldsymbol{S} \succeq 0$.
  Where $\succ$ and $\succeq$ stand for positive definite and positive semi-definite respectively.

Schur complement is of interest in this work since, together with the epigraph form, allow us to express a quadratic function as an LMI.

In this work, when we want to solve an equality task, we solve a problem expressed as $\min_{\boldsymbol{x}} . ||\boldsymbol{Cx} - \boldsymbol{d}||^2$. Using the epigraph form we can express it as

$$\begin{array}{ll} \min\limits_{\boldsymbol{x},\gamma} . & \gamma \\ \text{s.t.} & ||\boldsymbol{Cx} - \boldsymbol{d}||^2 - \gamma \leq 0 \end{array} . \tag{118}$$

Now, we can now modify the constraint $||\boldsymbol{Cx} - \boldsymbol{d}||^2 - \gamma \leq 0$, assigning $\boldsymbol{S}$ to be $S = \gamma - (\boldsymbol{Cx} - \boldsymbol{d})^\top (\boldsymbol{Cx} - \boldsymbol{d}) \geq 0$. Notice how here S is an scalar and, therefore, we can change the positive semidefinite symbol "$\succeq$" by "$\geq$". According to Schur complement, the constraint $S \geq 0$ will only hold if

$$\begin{bmatrix} \boldsymbol{I} & (\boldsymbol{Cx} - \boldsymbol{d})^\top \\ (\boldsymbol{Cx} - \boldsymbol{d}) & \gamma \end{bmatrix} \succeq \boldsymbol{0} , \tag{119}$$

which is an LMI and can be used as a constraint in the SDP problem.

B. 4-link planar manipulator

When researching and implementing new algorithms in robotics, it is crucial to provide a framework that permits to test the on-going developments in a fast and easy manner. This is the reason behind the creation of the work presented in this appendix. We have developed a Matlab based environment that allows you to test the *Hierarchical task control through Lexicographic SDP* presented in this work. It uses a simple 4-link planar manipulator to gain insight into tasks and how the hierarchy among them affects the general behavior of a robot.

The code has been made public in the repository at [28]. In order to simulate the kinematics of a planar manipulator it uses the Peter Corke *Robotics Toolbox* [29].

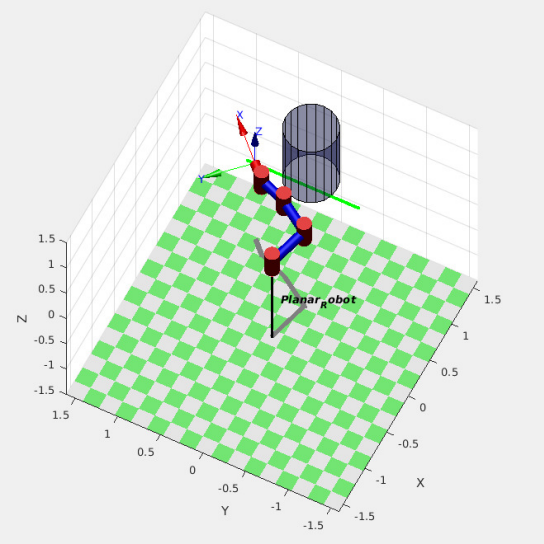
The aim of this appendix it is neither to explain the algorithm (detailed at Chapter 3) nor to detail the structure of the code, but to let know the reader its existence and possible uses. The algorithm generates a joint trajectory considering the following tasks: position tracking task, orientation tracking task, joint limits task as well as an obstacle avoidance task. We used the Matlab wrapper of the dedicated C++ solver used in the experiments with the real robot [27]. The code also contains the task plot functions that ease the comprehension of every task deployment. Implementing this kind of algorithm in such a simple robot can help in the comprehension of the tasks as well as to detect possible issues to be improved.

Even though it is in an early development stage, we tried to make a modular code so it can be useful for anyone who aims to broaden its functionalities. It offers a simple way to change task priorities, something useful to understand how hierarchical trajectory generator algorithms work. Number of links of the planar manipulator as well as dimension of every link can also be changed easily. Besides, tasks can be added easily typing the essential code. The last feature that is worth to mention is a desired path generator that is used along with the position and orientation tracking task. It generates uniformly accelerated trajectories between n given points.

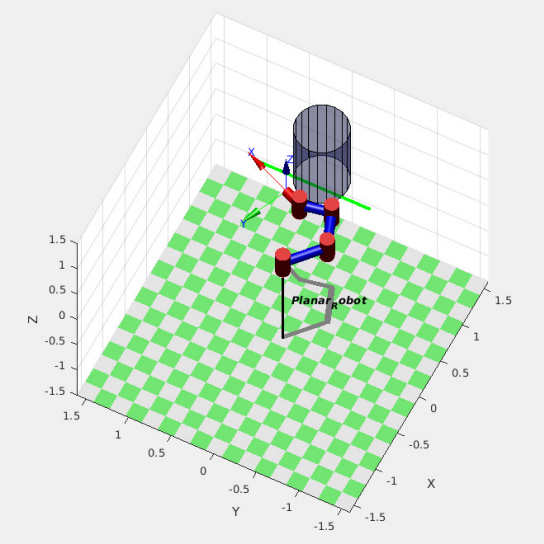
As a matter of example, we show in Figures 24 and 25 a set of images about two different simulations performed with the Matlab code. In both simulations the same tasks have been set. We have considered a position tracking task. In this case we try to follow a straight line with the end-effector (the green line in Figures 24 and 25). Besides, we want to follow this line keeping the fourth link aligned with the green line (orientation tracking task). As it can be seen in the representations, in the middle of the trajectory there is an

	Obs. Avo.	JL Pos.	Pos. Track.	Ori. Track.
Simulation 1	1	1	2	3
Simulation 2	1	1	3	2

Table 3: Priorities set depending on the simulations shown at Figures 24 and 25



(a) Giving up orientation tracking task.

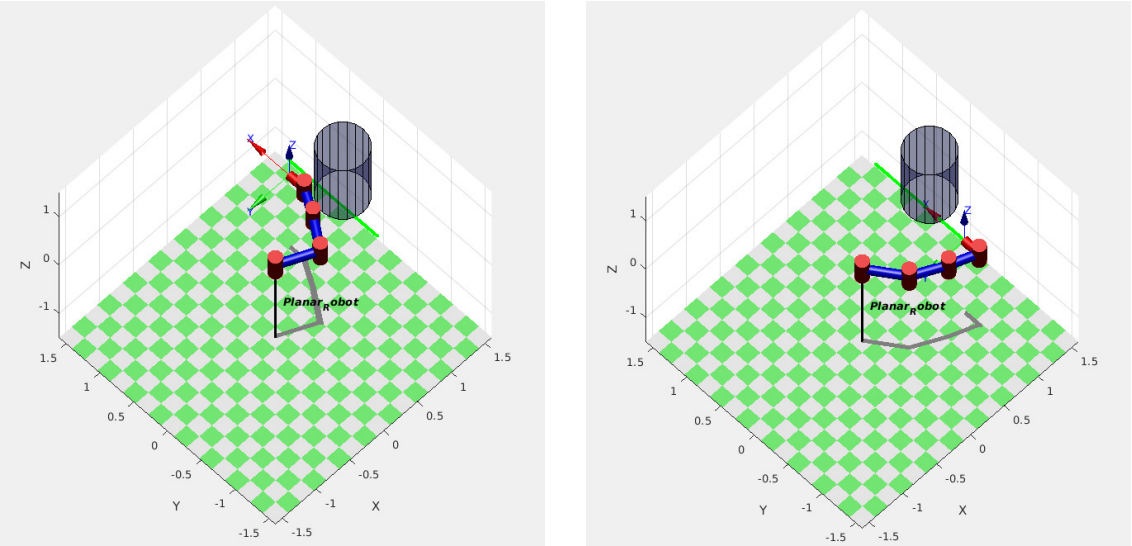


(b) Giving up orientation and position tracking task.

Figure 24: Simulation 1: 4-link planar manipulator with the position tracking task having higher priority than the orientation tracking task.

obstacle that should be avoided using an obstacle avoidance task. Last, we have also set a joint limits task. The difference between simulations are the priorities of the orientation tracking task and the position tracking task. These priorities are shown at Table 3.

As said before, this kind of implementations can be of use to check the usability of the algorithms developed. This can be used in order to get a less painful implementation in low-level programming languages for real robots. In Figure 24, the position tracking task has a higher priority than the orientation task. We can see that from the beginning (Figure 24a), the algorithm gives up the lowest priority task (orientation) due to the obstacle avoidance task. Instead, the position tracking task is kept. However, when the end-effector encounters the obstacle (Figure 24b), the position tracking task is also given up.



(a) Giving up position tracking task.

(b) Accomplishing with orientation and position tracking task.

Figure 25: Simulation 2: 4-link planar manipulator with the orientation tracking task having higher priority than the position tracking task.

Something different happens with the Simulation 2, shown at Figure 25. Here, the orientation can be kept during the whole trajectory sacrificing the position tracking task. This, can only be accomplished at the end of the trajectory (Figure 25b).