UNIVERSITY OF TRENTO

Department of Industrial Engineering
Master Degree in Mechatronics Engineering

A model predictive control approach for fatigue aware robotic heavy manipulation

Candidate:
Pasquale Buonocore

Supervisor:
Prof. Andrea Del Prete

Tutor:
Dott. Matteo Parigi Polverini
Dott. Arturo Laurenzi
Dott. Enrico Mingo Hoffman

Academic year 2019-2020

To my family...

Abstract

In the last decade disaster-response robotics has gained more and more attention in the robotic community for its potential benefit in supporting rescuers on disaster sites. For example, robots for disaster scenarios can assist human rescuers in performing heavy manipulation tasks in hazardous environments. A crucial problem for such robots, which is barely addressed in the literature, is the impact of thermal fatigue, which may deteriorate the task performance and eventually damage the robot. This thesis proposes to face the problem from a Nonlinear Model Predictive Control (NMPC) perspective, such that the motor thermal model is directly exploited to predict and constrain the motor temperature. The proposed approach is meant to react to and control the robot's thermal fatigue while executing a heavy manipulation task. Since NMPC is a technique based on the recursive solution of Optimal Control Problems (OCP), the first purpose of this thesis is to formalize an OCP describing a heavy object manipulation task for a dual arm manipulator, where thermal fatigue constraints are explicitly taken into account. Each instance of the NMPC is solved with the direct multiple shooting method, thanks to CasADi open-source library for optimization. Based on such optimal control problem, this work shows how the nonlinear model predictive control algorithm is implemented in the Robotic Operating System (ROS) environment. The control algorithm is finally tested on the CENTAURO platform by performing two heavy manipulation tasks, ensuring the prevention of thermal burnout. The first experiment consists in the platform holding a box in a certain position, while in the second one the robot is required to move the box center of mass along a circular trajectory at a fixed distance from its pelvis. We compared the NMPC results with the solution of the same problem without thermal constraints and the solution of a minimum effort problem. The proposed NMPC effectively prevented the thermal burnout by adapting the robot configuration at run-time, finding a trade-off between task completion and fulfillment of the thermal fatigue constraint.

Contents

List of Figures

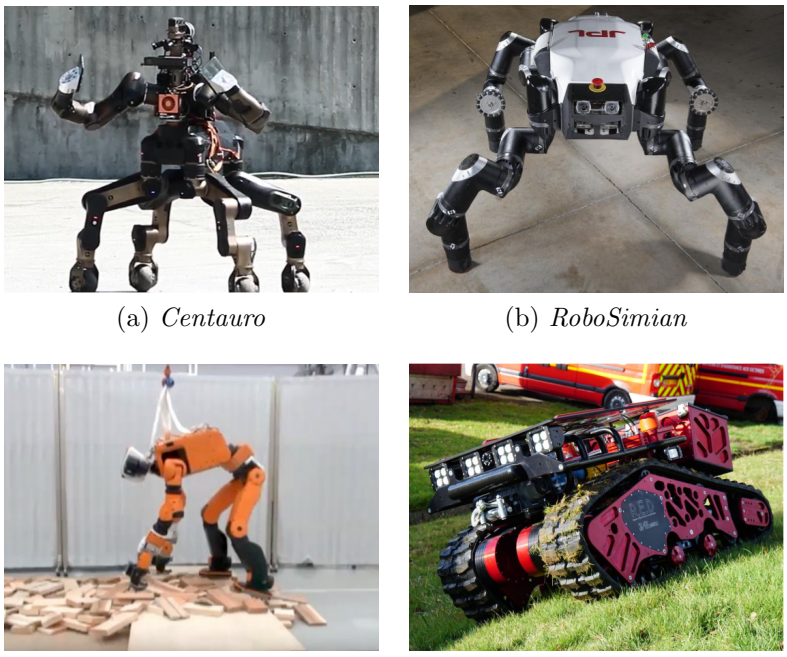
Chapter 1

Introduction

1.1 Disaster scenario robotics

From hurricanes to earthquakes, disaster scenarios clearly revealed the need for robots capable to meet real-world requirements. Humanoid robotic platforms have tremendous potential for this job and have attracted increasingly consideration in the last decade for the role in support of rescuers on disaster-response missions given that they can operate and assist in places too dangerous for humans accomplishing hazardous tasks.

Researchers and industries have been working intensely on the subject and in the last years, interesting results have been achieved. Figure 1.1 shows some solutions from all around the world with the purpose of delivering a robot capable of facing disaster response requirements. For instance, RoboSimian in Figure 1.1b is a limbed robot developed at *NASA's Jet Propulsion Laboratory* (JPL). It was designed to operate in environments too dangerous or difficult for human intervention and it is able to navigate difficult terrain and perform tasks requiring extreme dexterity. Figure 1.1c instead shows E2-DR, a strong and nimble human size robot which is the research result of the *Honda group*. It is capable of squeezing through 30 cm gaps, ramping stairs, stepladders and vertical ladders and so forth. Extremely positive results were obtained on the field with the Colossus robot designed and built by *Shark Robotics*. Colossus, in Figure 1.1d, is a versatile remote-controlled robot designed to support firefighters and first-responders in dangerous missions. With a high-pressure water cannon and powerful all-terrain treads, it can help extinguish fires, clear away debris, and evacuate victims. The Centauro robot, presented in figure 1.1a, was designed at the *Italian Institute of Technology* (IIT) with the purpose of accommodating the needs of real-world applications with high payload and harsh physical interaction demands like in disaster-response scenarios or heavy manipulation tasks. It combines powerful manipulation capabilities with a more reliable quadrupedal hybrid wheeled-legged locomotion concept.



(a) *Centauro* (b) *RoboSimian*

(c) $E2_DR$ (d) *Colossus*

Figure 1.1: Examples of disaster scenario robots

In disaster-response scenarios, robots should possess the capability to manipulate heavy payloads and interact with the damaged environment using heavy tools. Furthermore, they should also be able to protect themselves. Several safety-related aspects have been already addressed in robotics research, examples of which range from avoiding collisions [1] to controlling impact forces [2], [3]. Another important aspect of robot safety is related to actuator thermal burnout, which is described in the following.

1.2 The heating problem

One crucial problem with such robots performing demanding manipulation tasks is to keep actuator temperatures within safe limits, in order to avoid robot damage and contribute to the loss of productivity. For example, in electric motors, high temperature increases the winding resistance and causes demagnetization, which results in lower torque performance.

Those kinds of problems have already been faced in research. For example, the work on the tree-climbing robot RiSE [4] presents a control strategy that maximizes robot velocity while satisfying a constraint on the maximum permissible motor winding temperature. It uses thermal models for constrained robot control. The system uses a 3-lump thermal model which is highly robust and valid within a large range of environmental temperatures. It requires two temperature sensors to measure internal and external temperatures. Some ac-

tuators have only one internal temperature sensor which severely limits the approach. In this direction, researchers in [5] propose a technique for monitoring and management of robot fatigue, which is composed of a two-stage reaction process that is triggered by different levels of the joint temperature. In that case though, the management system is triggered once the joint temperature exceeds the bound. The paper [6] faces the robot fatigue problem as well. It does a time parameterization of pre-planned geometric robot paths using a time-optimization problem with a temperature constraint. Our problem is though of considerably higher complexity, both in the number of variables and of constraints: including not only temperature but also kinematic and collision. Other related research includes temperature prediction for humanoid robots [7], [8]. The robots used in that work (HRP-2, HRP-4R, STARO) cannot measure actuator winding temperature and the authors overcome this by assuming that the internal and external temperatures of the robot's actuators are the same, which is a severe assumption when we deal with heavy manipulation task. The work in [9] proposes a method to bring out the maximum performance of electric motors aggressively, and it uses a motor core temperature estimation to protect the motor burnout. The situation is different with respect to our since it plans to construct water cooling systems as method for motor cooling. The paper [10] proposes to deal with the optimization of velocity profiles of robots manipulator with a minimum time criterion subject to thermal constraints. A very similar method is presented in the paper [11], where researchers propose a method to reduce power consumption and temperature for robots with high-power DC actuators without cooling systems only through trajectory optimization motion planning. The study [12], instead, proposes a DSP-based approach to limit the current of the most stressed joint using a simple capacitance-resistance thermal model and a real-time calculation of the power losses. It is possible by taking advantage of the long thermal time constants.

1.2.1 Thermal models

To put a constraint on motor temperatures, a model describing the thermal time evolution needs to be identified. In literature, it is possible to find several thermal models ranging from simple to complex ones. In the paper [11], a thermal model for brushless DC motors is proposed. It uses a lumped capacitance method while considering heat transfer only through conduction and free convection. It uses two temperature sensors, which is not possible in our case as it will be shown in the next chapters. Papers [13], [14] presented two thermal model based on a finite element approach to analyze the motor thermal behaviour, whereas [15] presents a thermal network to describe the temperature distribution in an electric motor in steady-state and dynamic conditions. A simple approach instead is presented in the paper [9]. It presents a system of two equations to describe the temperature behaviour of a brushless motor, in order to propose a method to bring out the maximum performance of electric motors aggressively in humanoid robots.

1.3 Thesis contribution

The problem of overheating in robotics is usually tackled using cooling systems, but the development of an actuator cooler is not always possible due to either lack of funding, space, and design. A different approach to the problem could be the usage of control strategies to manage the fatigue of the robot. To face that challenge, this thesis proposes a *Model Predictive Control* (MPC) approach to react to and control the robot thermal fatigue while executing a heavy manipulation task. It protects from motor overheating by taking into account the motor temperature state in the optimization problem and using a thermal model to predict the motor's thermal behaviour. The main tasks of the thesis are listed as:

- Formalize the optimal control problem for fatigue aware robotic heavy manipulation.
- Formalize the nonlinear model predictive controller based on the previously defined optimal control problem.
- Implement the nonlinear model predictive control exploiting the robotic operating system.
- Test the model based controller on two different heavy payload manipulation tasks.

It is important to note that this method differs from a classical minimum effort approach because the torque reduction is temperature dependent and considers thermal fatigue of specific joints. For example, if some joints had considerably higher temperatures, torque reduction would be focused on those joints. On the contrary, the general effort reduction considers joints equally without taking their thermal fatigue into account. Our results show that a robot can perform high-load tasks while preventing motors from overheating during operation through joint reconfiguration; naturally, other joints must carry more load in order to produce the desired task in the Cartesian space. As a result, it is possible to extend the working time of the robot during tasks and avoid any potential damage due to the overheating of the actuators.

1.4 Overview

The remainder of the thesis is structured as follows:

In **Chapter 2** the background on nonlinear model predictive control is provided, starting from the concepts of nonlinear programming problems and optimal control problems. Successively, direct methods to deal with optimal control problems are described both from a theoretical and practical standpoint.

In **Chapter 3** background knowledge on mathematical modelling of robotic manipulators is provided, ranging from the definition of the end-effector pose to the analysis of the

forward and inverse kinematic and dynamics of robots.

In **Chapter 4** it is introduced the fixed-base bimanual manipulator platform over which the problem is formulated. Then it is formalized the optimal control problem describing a thermal bounded heavy payload manipulation in terms of optimization variables, cost function, and constraints. Finally, the complete optimal control problem is used as starting point to define the model predictive controller.

In **Chapter 5** details of the algorithm implementation are provided and the experimental results are summarized. Its first section gives the description of the real robot platform, CENTAURO, and the software framework (CasADi,Pinocchio, ROS) exploited in this work. Moreover, Chapter 5 presents the model predictive controller results in two heavy payload manipulation tasks.

In **Chapter 6** starting from the previous chapter results, conclusions are drawn and discussed together with possible future work.

Chapter 2

Nonlinear model predictive control

This chapter gives an accurate background of model predictive control. It starts with the concepts of nonlinear programming problem in Section 2.1 and optimal control problem in Section 2.2. Successively, the main techniques to deal with optimal control problems are introduced from both a theoretical and practical point of view. Section 2.3, instead, concludes the chapter with the nonlinear model predictive control theory.

2.1 Nonlinear programming

In order to discuss optimal control problems in Section 2.2, we need some background knowledge of *nonlinear programming* (NLP). NLP problems are an important class of continuous optimization problems and they can be seen as the process of solving an optimization problem where some constraints or the objective function are nonlinear. We formulate the nonlinear programming problem as:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & \mathbf{f}(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) = 0, \\ & \mathbf{h}(\mathbf{x}) \leq 0 \end{aligned} \tag{2.1}$$

where the *objective function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$, the *equality constraints* $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n_g}$ and the *inequality constraints* $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^{n_h}$ are assumed to be at least once continuously differentiable. A solution $\boldsymbol{x}^*$ is said to be feasible when it satisfies all the constraints. We define the *feasible set*:

$$\Omega := \{\mathbf{x} \in \mathbb{R}^n \quad | \quad \mathbf{g}(\mathbf{x}) = 0 \quad , \quad \mathbf{h}(\mathbf{x}) \leq 0\} \tag{2.2}$$

Among the feasible points, we define the optimal solution as the one that respects the first and second order optimality conditions, which are described next. Before that, we need to introduce the concept of *active constraint* and *active set*. An inequality constraint $\mathbf{h}_i(\mathbf{x}) \leq 0$ is said to be active at $\mathbf{x^*}$ if and only if $\mathbf{h}_i(\mathbf{x^*}) = 0$. The index set $\mathcal{A}(\mathbf{x}^*)$ containing the indices

of all the active constraints is defined as active set. Combining all the active inequalities with all equality constraints, we define the matrix $\bar{g}(\mathbf{x})$:

$$\tilde{\boldsymbol{g}}(\mathbf{x}) = \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{h}_i(\mathbf{x}) \, (i \in \mathcal{A}(\mathbf{x}^*)) \end{bmatrix} \tag{2.3}$$

We say that the *linear independence constraint qualification*(LICQ) holds at the feasible point if all the active constraints are linearly independent. This condition is equivalent to saying that the Jacobian matrix $\frac{\partial \bar{g}}{\partial x}(\mathbf{x}^*)$ is full rank. Now we can discuss the optimality conditions.

First order optimality conditions

In continuous optimization, a feasible point $\boldsymbol{x}^*$ can be considered a candidate local minimizer if it satisfies the first order optimality conditions. Once the linear independence constraint qualification is satisfied at $\boldsymbol{x}^*$, the feasible point is a local minimizer of (2.1) if there exist the so called multiplier vectors $\boldsymbol{\lambda} \in \mathbb{R}^{n_g}, \boldsymbol{\mu} \in \mathbb{R}^{n_h}$ such that the following system of equations is satisfied:

$$\begin{aligned} \nabla \mathbf{f}(\mathbf{x}^*) + \nabla \mathbf{g}(\mathbf{x}^*) \boldsymbol{\lambda}^* + \nabla \mathbf{h}(\mathbf{x}^*) \boldsymbol{\mu}^* &= 0 \\ \mathbf{g}(\mathbf{x}^*) &= 0 \\ \mathbf{h}(\mathbf{x}^*) &\leq 0 \\ \boldsymbol{\mu}^* &\geq 0 \\ \boldsymbol{\mu}_i^* \mathbf{h}_i(\mathbf{x}^*) &= 0, \quad i = 1, \ldots, n_h \end{aligned} \tag{2.4}$$

Note that the first optimality conditions are known in literature as the Karush–Kuhn–Tucker(KKT) conditions for constrained optimization. Those are an expansion of the Lagrange condition valid for only equality constrained problems. The first order optimality conditions are necessary conditions and they become sufficient to guarantee the global optimality when the problem is convex. When dealing with KKT conditions, it is important to define the *Lagrangian function* since it plays a crucial role in both convex and general nonlinear optimization:

$$\boldsymbol{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{f}(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{g}(\mathbf{x}) + \boldsymbol{\mu}^T \mathbf{h}(\mathbf{x}) \tag{2.5}$$

Using the Lagrangian it is possible to express the first KKT conditions as:

$$\boldsymbol{\nabla}_x \boldsymbol{\mathcal{L}}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0 \tag{2.6}$$

Second order optimality conditions

In general, the necessary conditions are not sufficient for optimality and additional information is required, such as the second order sufficient conditions. Consider a point $\mathbf{x}^*$ at which

linear independence constraint qualification holds together with multipliers $\boldsymbol{\lambda}^*$, $\boldsymbol{\mu}^*$ so that the KKT conditions are satisfied. Consider a basis matrix $\mathbf{Z} \in \mathbb{R}^{n \times (n - n_{\bar{\mathbf{g}}})}$ of the null space of $\frac{\partial \bar{g}}{\partial x}(\mathbf{x}^*) \in \mathbb{R}^{n_{\bar{g}} \times n}$, then the following conditions hold:

- If $\mathbf{x}^*$ is a local minimizer, then $\mathbf{Z}^T \boldsymbol{\nabla}_x^2 \boldsymbol{\mathcal{L}} (\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{Z} \geq 0$
- If $\mathbf{Z}^T \boldsymbol{\nabla}_x^2 \boldsymbol{\mathcal{L}} (\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \mathbf{Z} > 0$ then $\mathbf{x}^*$ is a local minimizer

The matrix $\boldsymbol{\nabla}_x^2 \boldsymbol{\mathcal{L}} (\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ plays an important role in optimization algorithms and is called the Hessian of the Lagrangian, while its projection on the null space of the Jacobian is called the reduced Hessian.

Nonlinear programming solver

To solve a nonlinear optimization problem with inequality constraints, two big families of methods exist, both aiming at solving the KKT conditions:

- **Interior point methods**(IP). It uses a penalty function as part of globalization strategy for Newton's method. The fundamental idea is to construct a function whose unconstrained minimum either is the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. An interesting introduction to barrier-method is given by [16]. From a practical point of view, there exist several open-source software, like the IPOPT package[17], that exploit this method to solve nonlinear programming problems.
- **Sequential quadratic programming methods**(SQP). It consists of simplifying the NLP (2.1) by using a quadratic approximation of the cost function and a linear approximation of the constraints, and then using the solution to the quadratic model to make a step towards a new point, where another quadratic model is formed. A very good discussion on the issues arising in SQP is given by Boggs and Tolle [18]. A lot of interesting implementations of the sequential quadratic approach can be found in literature, and qpOASES, WORHP and SNOPT are an example of them.

2.2 Continuous optimal control problem

A nonlinear programming problem is characterized by a finite set of optimization variables and constraints. Optimal control problems can involve continuous time functions as problem constraints, thus it can be seen as an infinite-dimensional extension of a nonlinear problem. Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. It usually includes a cost function that is a function of state and control variables. An optimal control describes the paths of the control variables that minimize the cost function. Once we define x(t) as the vector of differential state and u(t) as the vector of free control signals, an example of optimal control problem formulation may be given as:

$$
\begin{aligned}
&\min_{x(\cdot),u(\cdot)} \quad \int_0^T L(x(t),u(t))dt + E(x(T)) \\
&\text{s.t.} \\
&x(0) - x_0 = 0 && \text{(fixed initial value)}, \\
&\dot{x}(t) - f(x(t),u(t)) = 0, \quad t \in [0,T] && \text{(ODE model)}, \\
&h(x(t),u(t)) \leq 0, \quad t \in [0,T] && \text{(Path constraint)}, \\
&r(x(T)) \leq 0 && \text{(Terminal constraints)}
\end{aligned}
\tag{2.7}
$$

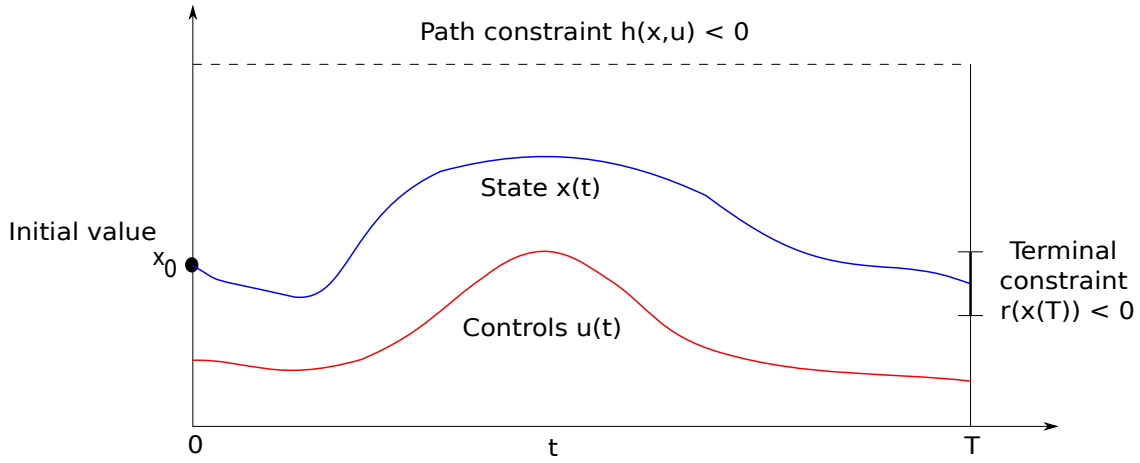


Figure 2.1: Optimal control time evolution

In (2.7), the cost function is composed of an integral contribution L(x,u) called the *Lagrange term*, and a terminal cost E(x(T)) called *Mayer term*. The combination of both is called a *Bolza objective*.

2.2.1 How to solve optimal control problems

Numerical methods for solving optimal control problem and more general variants emerged with the birth of the electronic computer in the 1950's and were in the beginning typically based on characterization on either the global optimum or the local optimum. There exist different methods for optimal control problems ranging from dynamic programming using the Bellman Optimality Equation, to indirect method using the Maximum Principle and direct methods. For a complete overview as well as detailed information on those methods see [19].

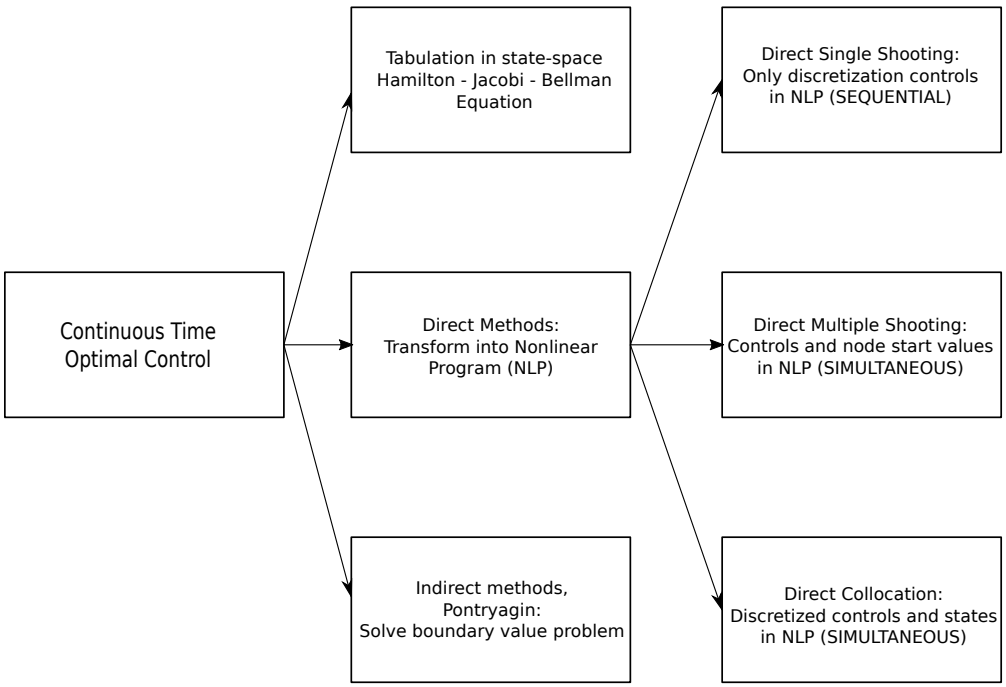


Figure 2.2: Optimal control methods

We will focus on direct methods. Those transform the original infinite optimal control problem into a finite dimensional nonlinear programming problem (NLP), which is then solved by exploiting numerical optimization methods. Roughly speaking, direct methods transform the continuous time dynamic system into a discrete time system and then proceed as described. The method is often sketched as "first discretize, then optimize". One of the most important advantages of direct compared to indirect methods is that they can easily treat inequality constraints. All direct methods are based on a finite dimensional parameterization of the control trajectory, but differ in the way the state trajectory is handled.

2.2.2 Direct approaches to continuous optimal control problem

For the solution of constrained optimal control problems in real world applications, direct methods are nowadays by far the most widespread and successfully used techniques, and are therefore used in this thesis. Two families of direct optimal control methods could be distinguished: sequential and simultaneous approach. In the *sequential approach*, represented by the direct single shooting method, existing software are used to eliminate the state trajectory from the problem formulation, leaving the control trajectory to be determined by the nonlinear programming solver. In the second approach, the *simultaneous approach*, the state trajectory is approximated by polynomials whose coefficients are determined together with the control trajectory in the nonlinear problem solver. A key advantage of the simultaneous approach is that it can handle unstable systems, where simulating the system might be impossible for the current guess of the control trajectory.

A hybrid approach is the direct multiple shooting method in which the state trajectory is only partially eliminated from the NLP. The direct multiple shooting method has some important properties of the simultaneous approach, in particular the handling of unstable systems and the suitability for parallel computations. At the same time, it avoids the need to store the whole state trajectory, which can be prohibitively expensive for large-scale systems.

Direct Single Shooting

The direct method eliminates the continuous time dynamic system in the problem. In order to reach the goal, we need to parametrize the control function *u(t)* by piece wise constant functions or by piecewise polynomials, as shown in Figure 2.3. We denote the finite control parameters by the vector q, and the resulting control function by *u(t; q)*.
The most widespread parameterization are piecewise constant controls, for which we choose a fixed grid $0 = t_0 < t_1 < ... < t_N = \mathrm{T}$, and N parameters $q_i \in R^{n_u}$, i = 0, ..., N-1, and then we set:

$$u(t; q) = q_i \ \forall \ t \in [t_i, t_{i+1}] \tag{2.8}$$

The dimension of the vector $q = (q_0, q_1, ..., q_{N-1})$ is $\mathrm{N}n_u$. Then the states vector $x(t)$ on [0, T] is regarded as dependent variables that are obtained by a forward integration of the dynamics, starting at x_0 and using the controls vector $u(t; q)$. We denote the resulting trajectory as $x(t; q)$. In order to discretize inequality path constraints, we choose a grid, typically the same as for the control discretization, at which we check the inequalities. Thus, in single shooting, we transcribe the optimal control problem into the following nonlinear programming problem:

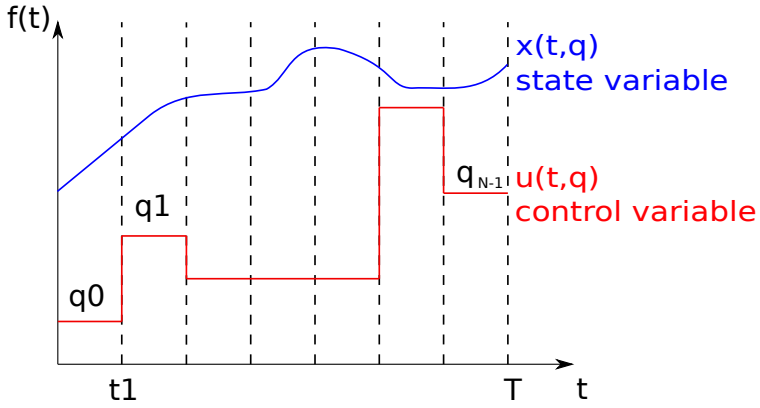


Figure 2.3: Direct single shooting method

$$\begin{aligned} \min_{q \in \mathbb{R}^{Nn_u}} \quad & \int_0^T L(x(t;q),u(t;q))dt + E(x(T;q)) \\ \text{s.t.} \quad & \\ h(x(t_i;q),u(t_i;q)) \leq 0 \quad & i = 0,...,N-1 \quad \text{(discretized path constraints)}, \\ r(x(T;q)) \leq 0 \quad & \quad \text{(discretized path constraints)} \end{aligned} \tag{2.9}$$

As the only variable of this NLP is the vector q $\in R^{Nnu}$ that influences all problem functions, the above problem can usually be solved by a dense NLP solver in a black-box fashion. In the case of piecewise controls we might use the fact that after the piecewise control discretization we have basically transformed the continuous time OCP into a discrete time OCP. This approach can handle an arbitrary number of path inequality constraints. Note that it has the same complexity that we obtain in the standard implementation of the multiple shooting approach, as explained next.

Direct Multiple Shooting

The direct multiple shooting method performs first a piecewise control discretization on a grid, exactly as we did in single shooting:

$$u(t;q) = q_i \text{ if } t \in [t_i, t_{i+1}] \tag{2.10}$$

Differently from the direct single shooting, the state variable $x(t)$ is not parametrized as function of the control, but it is solved separately on each interval $[t_i, t_{i+1}]$, starting with artificial initial values s_i, eq (2.11). We then need to ensure continuity between the intervals and we impose the additional constraint (2.12).

$$\dot{x_i} = f(x_i(t; s_i, q_i), q_i), t \in [t_i, t_{i+1}] \tag{2.11}$$
$$x_i(t_i; s_i, q_i) = s_i \tag{2.12}$$

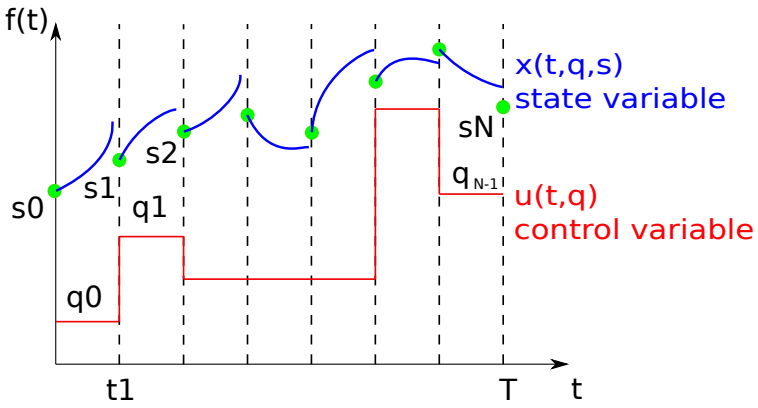


Figure 2.4: Direct multiple shooting method

Then, compute numerically the cost function integrals

$$l_i(s_i, q_i) := \int_{t_i}^{t_{i+1}} L(x_i(t; s_i, q_i), q_i)dt \tag{2.13}$$

In the end, the nonlinear problem that is solved in multiple shooting and that is visualized in the system of equations: (2.14)

$$\begin{aligned}
\min_{s,\,q} \quad & \sum_{i=0}^{N-1} l_i(s_i, q_i) + E(s_N) \\
\text{s.t.} \quad & x_0 - s_0 = 0 && \text{(Initial value)}, \\
& x_i(t_{i+1}; s_i, q_i) - s_{i+1} = 0, \quad i = 0, ..., N-1 && \text{(Continuity condition)}, \\
& h(s_i, q_i) \leq 0, \quad i = 0, ..., N && \text{(Discretized path constraints)}, \\
& r(s_N) \leq 0 && \text{(Terminal constraints)}
\end{aligned} \tag{2.14}$$

Moving from direct single shooting to direct multiple shooting we basically traded non linearity for problem size. The NLP in single shooting is small, but often highly nonlinear, whereas the NLP for multiple shooting is larger but sparser and typically less nonlinear. The direct collocation method goes a step further in the same direction, resulting in an even larger, but even sparser and possibly less nonlinear NLP.

Direct collocation

As we do in multiple shooting method, in direct collocation we discretize the infinite OCP in both controls and states on a fixed grid $0 = t_0 < t_1 < ... < t_N = \mathrm{T}$. The main difference between the two methods is that in each collocation interval, Figure 2.5 (recall that each collocation interval corresponds to an integrator step), the state is not obtained by integration but by approximation to a polynomial function.

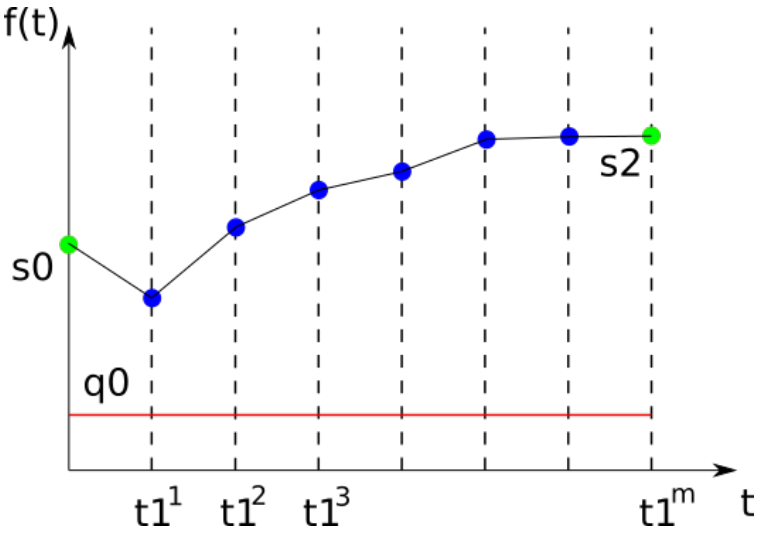


Figure 2.5: Direct collocation integration step

We choose a parameterization of the controls on the same grid with control parameters q_k that yield on each interval a function $u_k(t; q)$. On each collocation interval $[t_k,\ t_{k+1}]$, look at the example in figure 2.5, a set of m collocation points $t_k^0,\ t_k^1,\ .\ .\ .\ ,\ t_k^m$ is chosen and the trajectory is approximated by a polynomial $p_k(t; v_k)$ with coefficient vector v_k. As equalities of the optimization problem, we now require additional constraint, named collocation conditions, at each collocation point. We actually require that the polynomial derivative at each collocation point is the same as the dynamic system we approximate:

$$\begin{aligned} s_k &= p_k\left(t_k; v_k\right) \\ f\left(p_k\left(t_k^{(1)}; v_k\right), u_k\left(t_k^{(1)}; q_k\right)\right) &= p_k'\left(t_k^{(1)}; v\right) \\ &\vdots \\ f\left(p_k\left(t_k^{(m)}; v_k\right), u_k\left(t_k^{(m)}; q_k\right)\right) &= p_k'\left(t_k^{(m)}; v\right) \end{aligned} \tag{2.15}$$

If we summarize this system by the vector equation $c_k(s_k, v_k, q_k) = 0$ that has as many components as the vector v_i, if we require continuity across interval boundaries and we also approximate the cost function integral, we obtain a large scale and sparse non linear problem:

$$
\begin{aligned}
\min_{s,v,q} \quad & \sum_{0}^{N-1} l_k(s_k, v_k, q_k) + E(s_N) \\
\text{s.t.} \quad & s_0 - x_0 = 0 && \text{(Fixed initial value)}, \\
& c_k(s_k, v_k, q_k) \quad k = 0, ..., N-1 && \text{(Collocation condition)}, \\
& p_k(t_{k+1}; v_k) - s_{k+1} = 0 \quad k = 0, ..., N-1 && \text{(Continuity condition)}, \\
& h(s_k, q_k) \leq 0 \quad k = 0, ..., N-1 && \text{(Discretized path constraint)}, \\
& r(s_N)) \leq 0 && \text{(Terminal constraints)}
\end{aligned}
\tag{2.16}
$$

This large sparse NLP needs to be solved by structure exploiting solvers, and due to the fact that the problem functions are typically relatively cheap to evaluate compared with the cost of the linear algebra, nonlinear interior point methods are often the most efficient approach here.

2.3 Nonlinear Model Predictive Control

So far, we have analyzed one single optimal control problem and focused on ways to numerically solve this problem. Once we have computed such a solution, we might try to feed in open loop the corresponding real process with the obtained control trajectory. Unfortunately, the result will most probably be very dissatisfying, as the real process will typically not coincide exactly with the model that we have used for optimization.
Observing the real process during its time development will allow us to correct the control inputs online in order to get better performance. This procedure is called feedback control or closed-loop control. Even tough the idea of optimal feedback control via real-time optimization sounds challenging, it is common practice since decades in industrial process control under the name of *Model Predictive Control* (MPC). The name *nonlinear model predictive control*(NMPC), is reserved for the special case of MPC with underlying nonlinear dynamic systems, which leads typically to non-convex optimization problems.

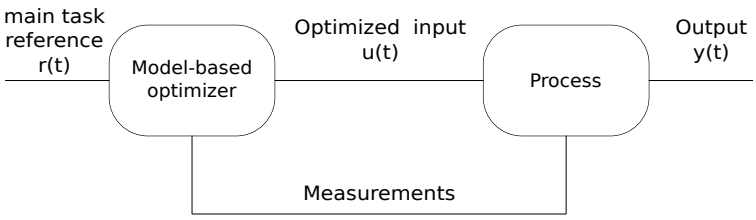


Figure 2.6: Model predictive control scheme

The idea of model predictive control is to solve an open-loop optimal control problem every time period since new state measurement/estimation is available. Every time period

an optimization problem is solved and the solution for the whole prediction horizon is found but only the first control action is applied to the plant. Next time a new measurement is obtained, another open-loop optimization problem is solved and so on. Therefore, it becomes a closed-loop control. The main stream implementation of MPC in discrete time, can be formulated as follows:

- observe the current state of the system $\mathbf{x}_0$
- predict and optimize the future behaviour of the process on a limited time window of N steps, called *prediction horizon*, by solving an open-loop optimization problem starting at the state $\mathbf{x}_0$
- implement the first control action $\mathbf{u}_0$ at the real process, move the optimization horizon one time step forward and repeat the procedure.

MPC is sometimes also called *receding horizon control* due to this movement of the prediction horizon.

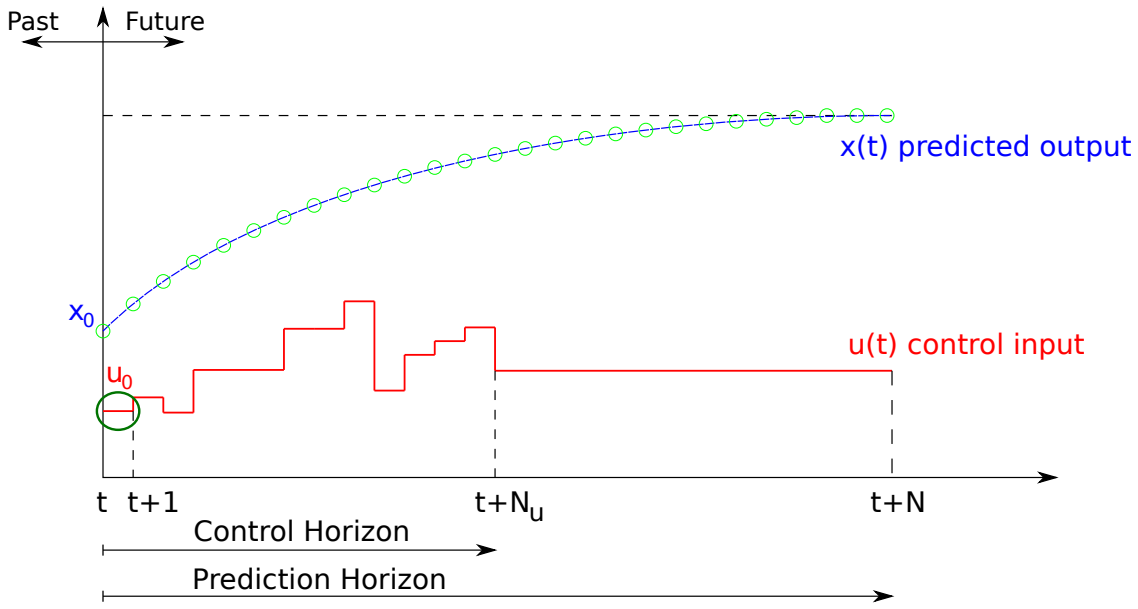


Figure 2.7: Receding horizon control

Model Predictive Control utilizes a built-in descriptive model of a system to predict the behavior of the system to the optimized control commands. These predictions are used to refine the optimization, anticipating future system responses. Moreover, it incorporates the ability to set constraints on the input and output values in the calculation of the control commands. Since MPC solves at each time sample an optimal control problem, it would be nice to reduce the computation time. A common technique, as shown in figure 2.7, consist in reducing the degree of freedom of the controller setting the *Control Horizon*. It means

considering constant the control input after a certain number of samples, considered that those will not influence the outcome as the first control steps do. This technique reduces the complexity of the problem and results in smaller computation time. Another important concept relative to model predictive control is the feasibility. We can ensure feasibility when only the input variable is constrained. Constraints on output variables may make the problem unfeasible and for $N < \infty$ there is no guarantee of feasibility, where $N = \infty$ ensures feasibility. The user must set a reasonable value of N based on the dynamic behaviour of the system to analyze. As far as stability is concerned, we may obtain it by setting and infinite prediction horizon or imposing a terminal constraint like:

$$x(t+N) = 0 \tag{2.17}$$

Chapter 3

Kinematics and Dynamics of a Robotic Manipulator

A robotic manipulator can be seen as a kinematic chain of rigid bodies connected through several types of joints. One end is generally fixed to a base, whereas an end-effector is fixed to the other end. In order to control an object, the study of the end-effector position and orientation is needed. This chapter starts with some mathematical background of rigid body pose in Section 3.1 and then it is focused on the main concept of kinematics and dynamics of manipulator in Sections 3.2,3.3.

3.1 Mathematical preliminaries

Robotic manipulation implies that parts and tools will be moved around in space by some sort of mechanism. This naturally leads to a need for representing positions and orientations of parts. Because of that, mathematical quantities that represent position and orientation need to be described together with additional entities.

3.1.1 Description of a pose

Once a world reference frame is defined, it is possible to identify any point in space with a [3x1] position vector. Because it is common to identify several coordinate systems in addition to the world one, vectors must be tagged with information identifying which coordinate system they are defined within. Figure (3.1) represents a rigid body, at which the reference frame {B} is attached, and a coordinate system, {A}, with three orthogonal unit vectors. The position of the rigid body is described by a vector and can equivalently be thought of as a position in space:

$$\mathbf{o}_B^A = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \tag{3.1}$$

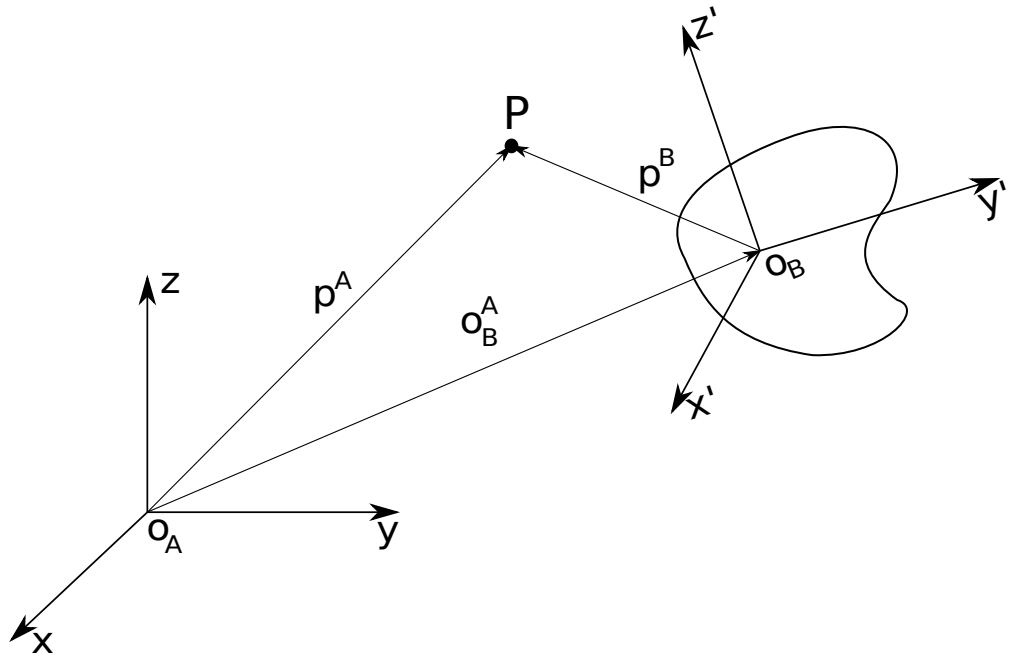


Figure 3.1: Pose of a rigid body

It is usually necessary to describe the orientation of a body in space in addition to its position. If the orientation is not given, the complete location of a body is still not totally specified since it may be oriented arbitrarily while keeping the same position. In order to describe the orientation of a body, it is common to attach a coordinate system to the body and then give a description of this coordinate system relative to the reference system using a matrix called rotation matrix, like:

$$\mathbf{R}_B^A = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{3.2}$$

This particular rotation matrix describes {B} relative to {A}, more precisely it describes the unit vectors of its three principal axes in terms of the coordinate system {A}. According to the literature [20], there are several ways to compose the rotation matrix, ranging from *euler angles, zyz angles, rpy angles, angle and axis* or *unit quaternions*.

Homogeneous transformation

As it was said in the previous paragraph, the pose of a rigid body is expressed with the position vector and a rotation matrix expressed in terms of the components of the unit vectors of a frame attached to the body.

Starting from the picture (3.1), consider the point P in space. The vector p^A is identified as the vector of coordinates of P with respect to the reference frame {A}. Consider now the frame {B}, whose origin is described by o_B^A in the fixed reference frame and whose rotation

is described by the rotation matrix R_B^A. Let also p^B be the vector of coordinates of P with respect to Frame {B}. According to simple geometry, the position of point P with respect to the reference frame {A} can be expressed as:

$$\mathbf{p}^A = \mathbf{o}_B^A + \mathbf{R}_B^A \mathbf{p}^B \tag{3.3}$$

Thus, (3.3) represents the coordinate transformation of p^B vector between two frames. The inverse transformation can be obtained by premultiplying both sides of (3.3) by R_B^{AT}; it follows that:

$$\mathbf{p}^B = -\mathbf{R}_A^B \mathbf{o}_B^A + \mathbf{R}_B^{AT} \mathbf{p}^A \tag{3.4}$$

If we adopt the homogeneous representation of a generic vector $\tilde{p} = [p\ 1]^T$, a more compact representation of the relationship between the coordinates of the same point in two different frames is possible thanks to the homogeneous transformation matrix.

$$\mathbf{A}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{o}_B^A \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.5}$$

As can be easily seen from (3.5), the transformation of a vector from Frame {B} to Frame {A} is expressed by a single matrix containing the rotation matrix of Frame {B} with respect to Frame {A} and the translation vector from the origin of Frame {A} to the origin of Frame {B}. Therefore, the coordinate transformation (3.3) can be compactly rewritten as:

$$\tilde{\boldsymbol{p}}^{\boldsymbol{A}} = \mathbf{A}_B^A \tilde{\boldsymbol{p}}^{\boldsymbol{B}} \tag{3.6}$$

The coordinate transformation between Frame {A} and Frame {B} is described by the homogeneous transformation matrix A_B^A which satisfies the equation:

$$\widetilde{\boldsymbol{p}}^B = \boldsymbol{A}_A^B \widetilde{\boldsymbol{p}}^A = \left(\boldsymbol{A}_B^A\right)^{-1} \widetilde{\boldsymbol{p}}^A \tag{3.7}$$

This matrix is expressed in a block-partitioned form as:

$$\mathbf{A}_A^B = \begin{bmatrix} \mathbf{R}_B^{AT} & -\mathbf{R}_B^{AT} \mathbf{o}_B^A \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_A^B & -\mathbf{R}_A^B \mathbf{o}_B^A \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.8}$$

3.1.2 Geometric Jacobian and Analytical Jacobian

The jacobian constitutes one of the most important tools for manipulator analysis. It is used to find singularities, to study redundancy, to determine inverse kinematics algorithms and to describe the mapping between forces applied to the end-effector and resulting torques at the joints. Above all, Jacobians describe the relationship between the joint velocities and the end-effector linear and angular velocities in a linear fashion. There exist two different formulations: *Geometrical jacobian* and *Analytical jacobian*.

Geometrical jacobian

The *geometrical jacobian* finds the relationship between the joint velocities and the end-effector linear and angular velocities and it is constructed according to a geometric technique in which the contributions of each joint velocity to the components of end-effector linear and angular velocity are taken into account. In other words, it is desired to express the end-effector linear velocity p_e and angular velocity ω_e as a function of the joint velocities $\dot{q}$, which depends on the manipulator configuration. With respect to the linear velocity, the geometrical jacobian is computed according to the formula:

$$\dot{\mathbf{p}}_e = \sum_{i=1}^{n} \frac{\partial \mathbf{p}_e}{\partial \mathbf{q}_i} \dot{\mathbf{q}}_i = \sum_{i=1}^{n} \boldsymbol{\jmath}_{P_i} \dot{\mathbf{q}}_i \tag{3.9}$$

whereas respect to the angular velocity:

$$\boldsymbol{\omega}_e = \sum_{i=1}^{n} \boldsymbol{\omega}_{i-1,i} = \sum_{i=1}^{n} \boldsymbol{\jmath}_{Oi} \dot{\mathbf{q}}_i \tag{3.10}$$

In a more compact formulation it is possible to summarize the concept identifying the so called *differential kinematics equation*, where $\mathbf{J}(\mathbf{q})$ is the geometrical jacobian.

$$\mathbf{v}_e = \begin{bmatrix} \dot{\mathbf{p}}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \boldsymbol{J}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} \mathbf{J}_p \\ \mathbf{J}_o \end{bmatrix} \dot{\mathbf{q}} \tag{3.11}$$

Analytical jacobian

If the end-effector pose is specified in terms of a minimal number of parameters, it is possible to compute the jacobian via differentiation of the direct kinematics function with respect to the joint variables.

$$\dot{\mathbf{p}}_e = \frac{\partial \mathbf{p}_e}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_P(\mathbf{q})\dot{\mathbf{q}} \tag{3.12}$$

$$\dot{\boldsymbol{\phi}}_e = \frac{\partial \boldsymbol{\phi}_e}{\partial \mathbf{q}} \dot{\mathbf{q}} = \mathbf{J}_\phi(q)\dot{\mathbf{q}} \tag{3.13}$$

The difference with respect to the geometrical jacobian regards the rotational speed. In fact, when the minimal representation of orientation [1] in terms of three variables ϕ_e is considered, its time derivative $\dot{\phi}_e$ in general differs from the angular velocity vector defined above. That it is nice to know and tell us that if we are interested in the end effector linear velocity, it is better to use the geometric jacobian.

[1] A minimal representation of orientation can be obtained by using a set of three independent angles $\Phi = [\phi\ \theta\ \psi]^T$.

$$\dot{\mathbf{x}}_e = \begin{bmatrix} \dot{\mathbf{p}}_e \\ \dot{\boldsymbol{\phi}}_e \end{bmatrix} = \begin{bmatrix} \mathbf{J}_P(\mathbf{q}) \\ \mathbf{J}_\phi(\mathbf{q}) \end{bmatrix} \dot{\mathbf{q}} = \mathbf{J}_A(\mathbf{q})\dot{\mathbf{q}} \tag{3.14}$$

For certain manipulator geometries, it is possible to establish a strong equivalence between $\mathbf{J}$ and $\mathbf{J}_A$. In fact, when the DOFs cause rotations of the end-effector all about the same fixed axis in space, the two Jacobians are essentially the same.

3.1.3 Singular configuration

As it is mentioned above, the Jacobian in the differential kinematics equation defines a linear mapping between the vector $\dot{\mathbf{q}}$ of joint velocities and the vector $\mathbf{v}_e = [\dot{\mathbf{p}}_e{}^T \boldsymbol{\omega}_e^T]^T$ of end effector velocity.

$$\mathbf{v}_e = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{3.15}$$

The jacobian is a function of the configuration $\mathbf{q}$ and the configurations at which $\mathbf{J}$ is rank-deficient are defined kinematic singularities. It is commonly of high interest to have knowledge of the singularities since at its neighborhood small velocities in the operational space may cause large velocities in the joint space, or because a robot in singularity condition shows reduced structure mobility. Moreover, such condition may induce infinite solutions to the inverse kinematics problem.

3.1.4 Redundancy

A manipulator is said to be redundant when it has a number of degrees of freedom which is greater than the number of variables that are necessary to describe a given task. A manipulator is intrinsically redundant when the dimension of the operational space is smaller than the dimension of the joint space. Redundancy is a concept relative to the task assigned to the manipulator. It contributes to robot dexterity and facilitates, enabling avoidance of mechanical limits of robot joints, obstacle avoidance, singularity avoidance, optimization of robot dynamics, etc. Unfortunately, redundancy usually increases the mathematical complexity of the robot control problem. Its implications are particularly pointed out in the inverse kinematics problem, as it is shown in the next paragraph.

3.2 Forward and inverse kinematics

3.2.1 Forward kinematics

Kinematics is the study of motion that deals with the subject without regard to the forces that cause it. The direct kinematic equation allows to express the end-effector pose with respect to the fixed reference frame as function of the joint variables. It is clear that the pose of a body with respect to a reference frame is described by the position vector of the origin and the unit vectors of a frame attached to the body. Thus, the direct kinematics function is described by the homogeneous transformation matrix:

$$\mathbf{A}_i^j(q) = \begin{bmatrix} \mathbf{n}_i^j(q) & \mathbf{s}_i^j(q) & \mathbf{a}_i^j(q) & \mathbf{p}_i^j(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

where q is the $[n \times 1]^T$ vector of joint variables, $\mathbf{n}_i^j, \mathbf{s}_i^j, \mathbf{a}_i^j$ are the unit vectors of a frame attached to the i^{th} body, and p_i is the position vector of the origin of such a body with respect to the origin of the base frame j.

3.2.2 Inverse kinematics

The relationship between the joint variables and the end effector position and orientation is described by the forward kinematics equation. On the other hand, the inverse kinematics problem consists of the determination of the joint variables corresponding to a given end-effector position and orientation. Even though the direct kinematics is quite simple to compute, the inverse kinematics problem is much more complex because the equation to solve are usually nonlinear, and thus it is not always possible to find a closed-form solution. A second problem is due to the fact that multiple solutions may exist or even infinite solution may exist in the case of redundant manipulators. Closed-form solutions, it might be appropriate to resort to numerical solution techniques; these clearly have the advantage of being applicable to any kinematic structure, but in general they do not allow computation of all admissible solutions.

In the following some approaches to solve the inverse kinematic problem are briefly introduced for both non redundant and redundant manipulators. When necessary in this thesis though, the inverse kinematic solution has been computed formulating the problem as non linear optimization problem, as shown next.

Differential inverse kinematic for non redundant manipulators

Geometrical jacobian finds a relationship between the joint velocities and the corresponding end-effector twist. That suggests the possibility to utilize the differential kinematics equation to face the inverse kinematics problem when we deal with non redundant manipulator with full-rank jacobian. The joint trajectory $\mathbf{q}$(t), $\dot{\mathbf{q}}$(t) that reproduces the given trajectory can

be resolved considering equation (3.15) for a non redundant manipulator. In fact, the joint velocities can be obtained via simple inversion of the jacobian matrix.

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\mathbf{v}_e \tag{3.17}$$

If the initial manipulator posture is known, joint positions can be computed by integrating velocities over time. The integration can be performed in discrete time by resorting to numerical techniques, like the Euler integration method.

Differential inverse kinematic for redundant manipulators

A redundant manipulator is described by a rectangular jacobian matrix, which has more columns than rows and determines infinite solutions to the inverse kinematic problem. A possible solution method is to formulate the problem as a linear constrained optimization problem [20] that minimizes the quadratic cost function of joint velocities:

$$g(\dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{W}\dot{\mathbf{q}} \tag{3.18}$$

where $\mathbf{W}$ is a square symmetric positive definite weighting matrix. The Lagrange multiplier method is a suitable approach to solve that problem. More detailed information for both methods discussed so far are available here [20].

Inverse kinematic as nonlinear optimization problem

The inverse kinematic problem becomes quite tricky when the manipulator has some degrees of redundancy. In that case, a good idea would be solving that problem writing it as a nonlinear optimization problem. We look for a manipulator configuration $\mathbf{q} \in \mathbb{R}^n$ that minimizes the cost function:

$$||\mathbf{e}_p|| = w_p||\mathbf{p}_E^O - \mathbf{p}_{ref}|| + w_R||\mathbf{R}_E^0 - \mathbf{R}_{ref}|| \tag{3.19}$$

where $\mathbf{p}_E^O$ and $\mathbf{R}_E^O$ are the end effector position vectors and rotation matrix obtained from the forward kinematics and function of the configuration vector $\mathbf{q}$, whereas $\mathbf{p}_{ref}$ and $\mathbf{R}_{ref}$ are the desired end effector position and rotation matrix. The norm of the rotation matrix is computed as:

$$||\mathbf{R}_E^0 - \mathbf{R}_{ref}|| = ||\mathbf{R}|| = \mathrm{tr}(RR) = \sum_{i,j} R_{i,j}R_{i,j} \tag{3.20}$$

The optimization problem to solve can be hence summarized as:

$$\begin{aligned} &\underset{\mathbf{q} \in R^n}{\text{minimize}} \quad ||\mathbf{e}_p|| \\ &\text{subject to} \\ &\quad \mathbf{q}_{lb} \leq \mathbf{q}_k \leq \mathbf{q}_{ub} \quad \text{(Joint limits)} \end{aligned} \tag{3.21}$$

3.3 Forward and inverse dynamics

Derivation of the dynamic model of a manipulator has an important role for motion simulation and analysis of manipulator structures. The analysis of the dynamic model can be helpful for the computation of the forces and torques required for the execution of typical motions. There are essentially two methods to compute the dynamic equation of motion of a manipulator, the first method is based on the *Lagrange formulation*, whereas the second method is based on the *Newton–Euler* formulation and yields the model in a recursive form; it is computationally more efficient since it exploits the typically open structure of the manipulator kinematic chain. Both Lagrange formulation and Newton–Euler formulation allow the computation of the relationship between the joint torques and the motion of the structure:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{J}^T\boldsymbol{W} = \boldsymbol{\tau} \tag{3.22}$$

where $\boldsymbol{\tau}$ is a vector of robot joint torques, $\mathbf{q}$ is a vector of joint angles, $\mathbf{J}$ is Jacobian matrix of robotic arm, $\mathbf{M}$ is mass matrix, $\mathbf{C}$ is Coriolis and centrifugal matrix, $\mathbf{g}$ is gravity vector and $\mathbf{W}$ is the force that the robot exerts on the environment.

In the study of dynamics, it is relevant to find a solution to two kinds of problems concerning computation of direct dynamics and inverse dynamics. The forward dynamics problem consists of determining, for $\mathrm{t} > t_0$, the joint accelerations $\ddot{\mathbf{q}}(\mathrm{t})$ resulting from the given joint torques $\boldsymbol{\tau}(\mathrm{t})$ and the possible end-effector forces $\mathbf{W}(\mathrm{t})$. The inverse dynamics problem consists of determining the joint torques $\boldsymbol{\tau}(\mathrm{t})$ which are needed to generate the motion specified by the joint accelerations $\ddot{\mathbf{q}}(\mathrm{t})$, velocities $\dot{\mathbf{q}}(\mathrm{t})$, and positions $\mathbf{q}(\mathrm{t})$, once the possible end-effector forces $\mathbf{W}(\mathrm{t})$ are known.

3.4 Statics

The goal of statics is to determine the relationship between the generalized forces applied to the end-effector and the generalized forces applied to the joints with the manipulator at an equilibrium configuration. Let $\boldsymbol{\tau}$ denote the vector of joint torques and $\mathbf{W}$ the vector of end-effector forces, a manipulator is in equilibrium configuration when the joint accelerations $\ddot{\mathbf{q}}$ and joint velocities $\dot{\mathbf{q}}$ are zero. Substituting that condition in equations (3.22), we obtain:

$$\boldsymbol{\tau} = \mathbf{J}^T(\boldsymbol{q})\mathbf{W} + \mathbf{g}(\mathbf{q}) \tag{3.23}$$

stating that the relationship between the end-effector forces and the joint torques is established by the transpose of the manipulator geometric Jacobian.

Chapter 4

Fatigue aware NMPC for bimanual heavy manipulation

This chapter aims at formalizing the optimal control problem for fatigue aware robotic heavy manipulation. Section 4.1 describes the fixed-base bi-manual manipulator model utilized in the optimization as well as the choice of state and control variables. Section 4.2 introduces the choice of optimization variables that are optimized by the proposed solver in order to minimize the cost function described in Section 4.3 and to respect the constraints presented in Section 4.4. Finally, in Section 4.6 the complete optimal control problem is exploited to define the model predictive controller.

4.1 Problem statement

In Chapter 1, we discussed the benefits of human-centred robots in role of support in disaster-response missions and we introduced one of the problems those platforms are subject to: the thermal fatigue. In order to address this problem in the context of a heavy manipulation task, this work proposes to use an optimal control approach. Our target platform is a fixed base dual-arm manipulator as in Figure 4.1. In the following, the system dynamic and all the main components that characterize the optimal control problem are introduced. Its solution will produce a physical consistent and thermal bounded manipulation.

4.1.1 Dual arm manipulator dynamic model

The dynamic model allows to compute the rate of change of the system state given its current state value $\mathbf{x}(t) \in \mathbb{R}^n$ plus the value of the control input $\mathbf{u}(t) \in \mathbb{R}^k$. This relationship is modeled with an ordinary differential equation (ODE):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.1}$$

Since this relation depends on the state variable $\mathbf{x}$(t) and on the physical quantity considered as input $\mathbf{u}$(t), it is necessary to discuss them.

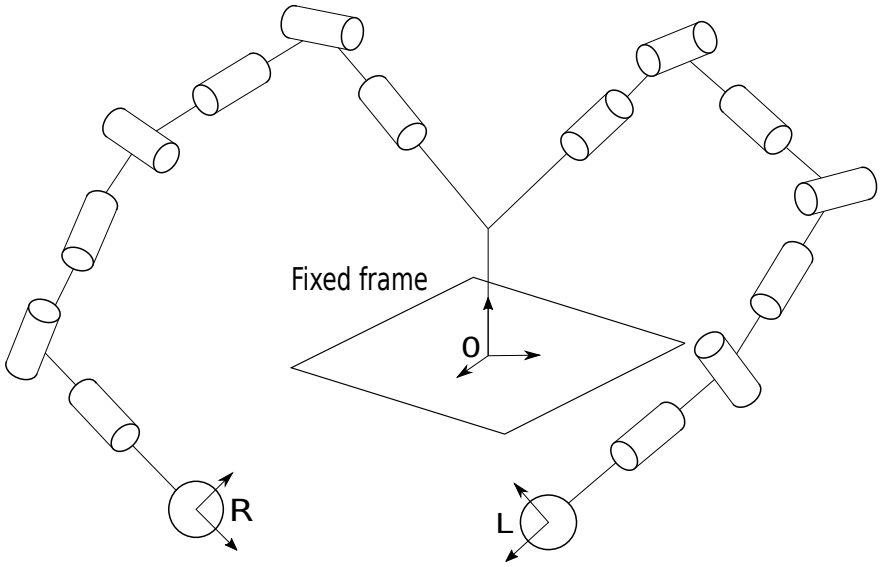


Figure 4.1: Bimanual manipulator model

The left and right arm configuration of the bimanual manipulator in Figure 4.1 are respectively described by the joint angle vectors $\mathbf{q}_L \in \mathbb{R}^{n_L}$ and $\mathbf{q}_R \in \mathbb{R}^{n_R}$. Therefore, the system can be completely described through the generalized coordinates $\mathbf{q} \in \mathbb{R}^n$, where $n = n_L + n_R$.

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_L \\ \mathbf{q}_R \end{bmatrix} \tag{4.2}$$

Considering the dual arm manipulator as a collection of rigid bodies connected through actuated joints, we can describe the relationship between the torque $\boldsymbol{\tau}(t) \in \mathbb{R}^n$ acting at each joint and the corresponding motion thanks to the equation:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{J}^T\boldsymbol{W} = \boldsymbol{\tau} \tag{4.3}$$

where $\mathbf{J} \in \mathbb{R}^{2m \times n}$ is the geometric Jacobian matrix of the manipulators and m=6 in that specific case. The symmetric positive definite $\mathbf{M} \in \mathbb{R}^{n \times n}$ is the inertia matrix. $\mathbf{C} \in \mathbb{R}^{n \times n}$ is the Coriolis and centrifugal matrix, $\mathbf{g} \in \mathbb{R}^n$ is the vector of torques due to gravity and $\mathbf{W} = [\mathbf{W}_L \mathbf{W}_R]^T \in \mathbb{R}^{2m}$ is the wrench vector of forces and torques the robot exerts on the environment. From (4.3), a more simplified dynamic representation can be obtained considering a *quasi-static* assumption (4.4). Quasi-static means that at a given instant in time we assume the problem to be static. It works well when inertial effects are very low:

$$\ddot{\mathbf{q}}(t) \approx 0 \tag{4.4}$$

This assumption is needed to simplify as much as possible the system dynamics which is necessary since the computation time plays a critical role in MPC problems. In fact, the MPC solves at each time stamp an optimal control problem and that should be done in as

little time as possible. There is also another reason why that assumption can be accepted, and it is related to the actuator thermal dynamics. The actuators thermal dynamics, in fact, is generally much slower than the robot dynamics and it can be considered predominant. Thus, assuming that (4.4) holds, the equation (4.3), becomes:

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{J}^T\boldsymbol{W} = \boldsymbol{\tau} \tag{4.5}$$

4.1.2 State and control variables

In order to take into account the robot thermal fatigue, to fully describe the double manipulator state we need to monitor also the joint temperatures evolution in time. That suggests to define the state variable $\mathbf{x}(\mathrm{t}) \in \mathbb{R}^{2n}$ as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \mathbf{T} \end{bmatrix} \tag{4.6}$$

where $\mathbf{T}(\mathrm{t}) \in \mathbb{R}^n$ is the joint temperature vector. As far as the control variable is concerned, since the (4.4) holds, a convenient choice is to control the dual arm manipulator platform in joint angle velocity $\dot{\mathbf{q}}$. Thus, we define:

$$\mathbf{u} = \dot{\mathbf{q}} \tag{4.7}$$

4.2 Optimization variables

The optimal control solution minimizes the objective function, which is introduced in Section 4.3, optimizing over a vector of optimization variables. In this section, we identify that set of variables that depends on the method utilized to solve the optimal control problem. Among the three different methods introduced in Chapter 2, this thesis utilizes the direct multiple shooting method which discretizes both control and state of the system. Because of that, and since the bimanual manipulator system in Figure 4.1 under the quasi-static approach is controlled in joint angle velocity, out first optimization variables are the state vector $\mathbf{x}(\mathrm{t})$ and the joint velocity vector $\dot{\mathbf{q}}(t)$. Another optimization variable is the wrench vector at each contact point, where the manipulator exchanges forces and torques with the environment, as:

$$\mathbf{W}_i = \begin{bmatrix} F_{i_x} \\ F_{i_y} \\ F_{i_z} \\ M_{i_x} \\ M_{i_y} \\ M_{i_z} \end{bmatrix} \quad \text{for } i \in \{L, R\} \tag{4.8}$$

In this study though, we considered the robot capable of exchanging only forces at the contact points, which are fixed on the payload and defined as global variable in the fixed frame. Therefore, we introduce in the optimization vector the force components of each contact point, the first one being the left end-effector and the second one, the right end-effector.

$$\mathbf{F}_i = \begin{bmatrix} F_{i_x} \\ F_{i_y} \\ F_{i_z} \end{bmatrix} \text{ for } i \in \{L, R\} \tag{4.9}$$

To conclude, it is possible to identify the optimization variable vector as:

$$\mathbf{w} = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \\ \mathbf{F}_L(t) \\ \mathbf{F}_R(t) \end{bmatrix} \tag{4.10}$$

4.3 Cost function formulation

The cost function is the measure of the process behavior over the prediction horizon that is minimized with respect to the optimization variables. It is composed of a sum of tasks $\mathbf{Y}_i$, each one weighted with w_i according to its relevance in the optimization problem. In addition to that, it is a common practice to add regularization terms in the cost function. In mathematics, regularization is the process of adding information in order to solve an ill-posed problem. Moreover, with regularization we try to impose a certain behaviour to the final solution. Therefore, regularization terms on the joint velocities and on the forces at the contact are introduces in the cost function. One important point to consider is that the cost function weights are of high relevance, since those influence the solver in finding the optimal solution. An example of cost function at each time step of the multiple shooting method is:

$$\mathbf{J} = \sum_{i=0}^{n} w_i \mathbf{Y}_i(\mathbf{q}, \dot{\mathbf{q}}) + w_q(\dot{\mathbf{q}}^T \dot{\mathbf{q}}) + w_{LF}(\mathbf{F}_L^T \mathbf{F}_L) + + w_{RF}(\mathbf{F}_R^T \mathbf{F}_R) \tag{4.11}$$

4.4 Constraint formulation

In the previous Sections 4.2 and 4.3, we introduced the optimization variables and we discussed the cost function. In order to complete the optimal control problem formulation, we need to go through the problem constraints. The constraints are needed to both ensure a physically consistent robot behaviour and to respect actuation and structure limits.

4.4.1 State and control variable limits

Recall that the control variable is the joint angle velocity $\dot{\mathbf{q}}$(t) and the state vector is represented by the joint angles $\mathbf{q}$(t) and joint temperature $\mathbf{T}$(t). The joint angles and joint

velocities have to be consistent with the robot capabilities in order to allow a feasible and safe optimal trajectory. Thus, the first set of hard constraints that must be satisfied at each multiple shooting node k is:

$$\mathbf{q}_{LB} \leq \mathbf{q}_k(t) \leq \mathbf{q}_{UB} \tag{4.12}$$

$$\dot{\mathbf{q}}_{LB} \leq \dot{\mathbf{q}}_k(t) \leq \dot{\mathbf{q}}_{UB} \tag{4.13}$$

The temperature constraint is also mandatory to impose since we want to handle the robot's thermal fatigue. It is treated in the following.

4.4.2 Torque limits

According to the quasi-static assumption described at the beginning of the chapter, at each time step the motor torque can be computed as:

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{J}^T\boldsymbol{W} = \boldsymbol{\tau} \tag{4.14}$$

where we recall that $\mathbf{J} \in \mathbb{R}^{2mn}$ is the geometric Jacobian matrix of the dual arm manipulator, $\mathbf{C} \in \mathbb{R}^{n \times n}$ is Coriolis and centrifugal matrix, $\mathbf{g} \in \mathbb{R}^n$ is gravity vector and $\mathbf{W} \in \mathbb{R}^{2m}$ is the force that the robot exerts on the environment. In order to satisfy the limit imposed by the actuators, the optimized resulting trajectory needs to satisfy torque constraints at each multiple shooting node k, which are formalized as follows:

$$\boldsymbol{\tau}_{LB} \leq \boldsymbol{\tau}_k(t) \leq \boldsymbol{\tau}_{UB} \tag{4.15}$$

4.4.3 Joint temperature constraints

Model predictive control uses a model of the process to predict the future evolution of the motor thermal behaviour over some finite horizon. Because of that, one of the objectives of this thesis is to find a simple motor thermal model to use to constrain the maximum allowed joint temperature. Thanks to its simplicity, we employ the simple *brushless motor* (BLDC) thermal model described in [**model˙2**].

Thermal model

The thermal behaviour of the motor can be represented by a circuit similar to an electric one. That is possible thanks to the analogy between the two fields. In the brushless servo motor, the heat is mainly created from the stator and the thermal behaviours of the rotor can be ignored. In Figure 4.2 the BLDC thermal model is shown. The point W represents the winding, whose too high temperature determines motor failure, and S represents the iron of the stator. The capacitor C_θ represents the thermal capacity of the stator. A resistance $R_{\theta 0}$ represents the thermal transfer across the winding insulation to the iron. The transfer

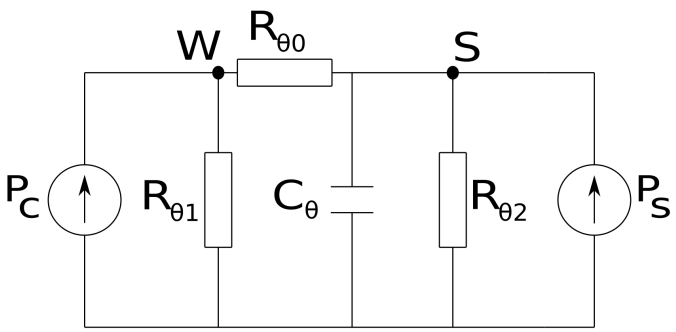


Figure 4.2: BLCD - Equivalent electric circuit

of heat between stator and rotor is represented by a resistance $R_{\theta 1}$. The transfer of heat to the surroundings is represented by a resistance $R_{\theta 2}$. In this analogue, the temperature is represented by the voltage and the thermal power loss by a current source, as it is explained in the next section. The motor winding temperature formula can be written:

$$T_w = \frac{R_{\theta 1} R_{\theta 2}}{R_{\theta 1} + R_{\theta 2}}(P_c + P_s) - \frac{R_{\theta 1} R_{\theta 2}}{R_{\theta 1} + R_{\theta 2}} C_\theta \frac{dT_w}{dt} \tag{4.16}$$

where T_w is the winding temperature. $R_{\theta 0}$ is ignored. The thermal resistances and the thermal time constant are normally given by the manufacturer or they need an identification, as it is shown in Appendix. The thermal time constant denote by t_θ is R_θ C_θ. The discrete-time solution for equation (4.16) can be expressed as:

$$T_{k+1} = e^{\frac{-h}{t_\theta}} T_k + P_{\theta k} R_\theta (1 - e^{\frac{-h}{t_\theta}}) \tag{4.17}$$

where h is the sample time and P_θ is the total thermal loss in a brushless motor. In the next paragraph, a method to evaluate the thermal power loss of a brushless motor is shown.

Thermal power

To use the model described above, a method to compute the thermal power generated inside the motor is needed. The thermal power must be calculated in real time in order to predict the transient temperature. The major loss in the brushless servo motor is given by the copper loss which is $P_c = R_a I_a^2$. Other losses include the windage and mechanical losses and eddy-current losses, which can be approximated as proportional to the square of the speed. Hence a resistance denoted by R_h can be placed in the motor equivalent circuit as shown in Figure 4.3. In this circuit analogy, the velocity is represented by the back EMF, E. The power loss can be obtained as:

$$P_s = \frac{E^2}{R_h} \tag{4.18}$$

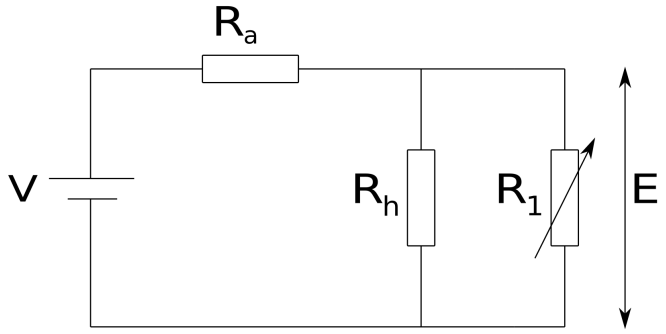


Figure 4.3: BLCD - Thermal power circuit

Since the motor current is strictly related to the motor torque τ through a motor constant K_m, and the rotational speed is $\dot{q}$, the total thermal power loss of the brushless motor can be computed as function of the optimization variables defined above:

$$P_\theta = P_c + P_s = R_a I_a^2 + \frac{E^2}{R_h} = R_a \frac{\tau^2}{K_m^2} + \frac{\dot{q}^2}{R_h} \tag{4.19}$$

where P_c is copper loss, P_s is the losses related to speed, R_h is the equivalent thermal loss resistance, R_a is the equivalent DC armature resistance, I_a is the equivalent DC armature current and R_1 is an equivalent resistance representing the variable mechanical load. Substituting eq.(4.19) in eq.(4.17), it is possible to define the thermal model that will be used to impose the thermal constraint:

$$T_{k+1} = e^{\frac{-h}{t_\theta}} T_k + (R_a \frac{\tau_k^2}{K_m^2} + \frac{\dot{q}_k^2}{R_h}) R_\theta (1 - e^{\frac{-h}{t_\theta}}) \tag{4.20}$$

Considering that the joint torques are computed through the eq.(4.14) and are function of q(t) and $\dot{q}$(t), the temperature integration, eq.(4.20) at the next step, can be written as:

$$T_{k+1} = f_{int}(q_k, \dot{q}_k, T_k) \tag{4.21}$$

Then at each time sample this formula is used to predict the motor power consumption and above that temperature constraint can be defined:

$$T_{k+1} \leq T_{bound} \tag{4.22}$$

4.4.4 End-effector constraints

To guarantee a correct payload manipulation we need to ensure a constant end-effectors relative pose. This constraint allows the double arm manipulator to grasp the payload for all task execution, avoiding its fall. In this work two methods are proposed to impose such constraint:

- **Relative pose constraint**. These constraints impose a fixed orientation and end-effector relative position during all the manipulation task.
- **Relative twist constraint**. Those constraints impose a zero relative twist (linear and angular relative velocity) between the two end-effectors during all the manipulation task.

Figure 4.4 shows a dual arm manipulator platform and the necessary mathematical quantities to define the aforementioned constraint. It is possible to identify three reference frames: the fixed world reference frame {O}, the left end-effector reference frame {L} and the right end-effector reference frame {R}.

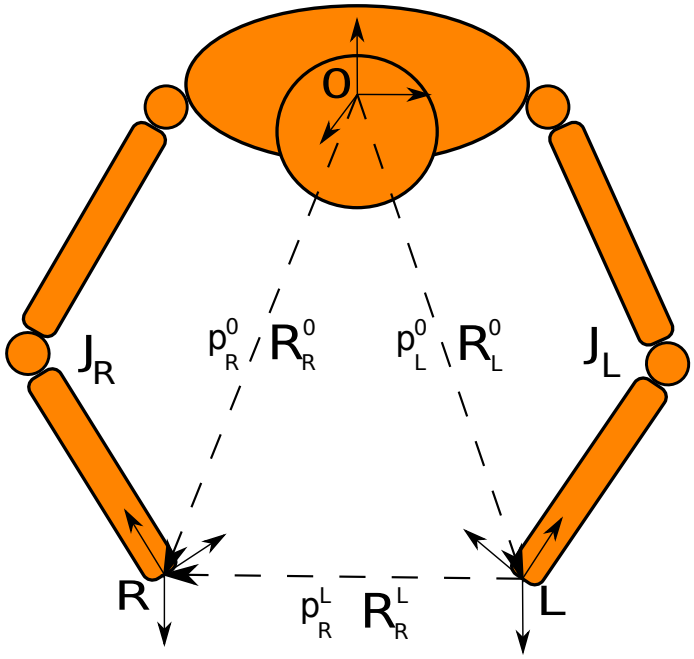


Figure 4.4: Dual arm manipulator platform

RELATIVE POSE CONSTRAINT

A relative pose constraint aims at ensuring a constant end-effectors distance and a fixed relative orientation. Thus, we split the relative pose constraint in relative position constraint and relative orientation constraint.

Relative position

In order to fix a relative position between end effectors, it is possible to require that the vector $\mathbf{p}_L^R$, which represents the position of the left end-effector in the right end-effector reference

frame, constant for all the manipulation period. Given the left end effector position in world frame $\mathbf{p}_L^0$, it is possible to express it in the right end effector reference frame according to the homogeneous transformation matrix $\mathbf{T}_0^R$:

$$\mathbf{p}_L^R = \mathbf{T}_0^R \, \mathbf{p}_L^0 \tag{4.23}$$

where the homogeneous transformation matrix:

$$\mathbf{T}_0^R = \begin{bmatrix} (\mathbf{R}_R^0)^T & -(\mathbf{R}_R^0)^T \, \mathbf{p}_R^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4.24}$$

Since we want such position to be constant during the problem evolution, we impose its derivative equal to zero. In discretized form, we assess the following constraint:

$$\mathbf{p}_{L\,k+1}^R - \mathbf{p}_{L\,k}^R = 0 \tag{4.25}$$

Relative orientation

Once the relative position is imposed as (4.25), we need something that ensures the end-effector relative orientation does not change while the robot manoeuvres the payload. In literature, it is common to define the orientation error between two frames once the rotation matrices $\mathbf{R}_1$ $\mathbf{R}_2$ are given. The orientation error can be computed as:

$$\mathbf{e}_o = [\mathbf{S}(\mathbf{R}_1 \, \mathbf{R}_2^T)]^{\mathbf{v}} \tag{4.26}$$

where $\mathbf{S}()$ represents the skew operator and $[]^{\mathbf{v}}$ is the operator that extracts the three components from the skew matrix. By definition, it is possible to compute the skew matrix of a general matrix $\mathbf{R}$ as:

$$\mathbf{S}(\mathbf{R}) = \frac{\mathbf{R} - \mathbf{R}^T}{2} \tag{4.27}$$

In our case, the orientation task can be fulfilled given the two end effectors rotation matrix $\mathbf{R}_L^0$, $\mathbf{R}_R^0$. We first compute the initial orientation error $\mathbf{e}_o(0)$ according to 4.26. Then, and at each time stamp k of the discretized time evolution, we compute the configuration dependent orientation error $\mathbf{e}_o(k)$ and impose:

$$\mathbf{e}_{ok} - \mathbf{e}_{o0} = 0 \tag{4.28}$$

RELATIVE TWIST CONSTRAINT

There is a second method to impose relative position and orientation constraint and this is obtained by imposing relative twist between end-effectors equal to zero.

$$\mathbf{J}_r \dot{\mathbf{q}} = 0 \tag{4.29}$$

The relative linear and angular velocity can be imposed thanks to the relative jacobian, $\mathbf{J}_r$, as it is explained in [21]. Figure 4.4 shows a dual-arm robot with its corresponding

reference frames {0}, {L}, {R} and Jacobians $\mathbf{J}_L$, $\mathbf{J}_R$ used in the derivation of the relative Jacobian. In fact, as it is explained below, the relative Jacobian can be expressed in terms of the individual manipulator Jacobians.

It is possible to express the relative linear and angular velocities between the two end-effectors as:

$$\begin{bmatrix} \dot{\mathbf{p}}_R^L \\ \dot{\boldsymbol{\omega}}_R^L \end{bmatrix} = \begin{bmatrix} -\mathbf{R}_0^L \mathbf{J}_{pL} \dot{\mathbf{q}}_L + \mathbf{S}(\mathbf{p}_R^L)\, \mathbf{R}_0^L \mathbf{J}_{oL} \dot{\mathbf{q}}_L + \mathbf{R}_0^R \mathbf{J}_{pR} \dot{\mathbf{q}}_R \\ \mathbf{R}_0^L \mathbf{J}_{oL} \dot{\mathbf{q}}_L + \mathbf{R}_0^R \mathbf{J}_{oR} \dot{\mathbf{q}}_R \end{bmatrix} \tag{4.30}$$

where:

- $\mathbf{R}_i^j$ is the rotation matrix of frame i in frame j
- $\mathbf{J}_{pi}$ represents the first three rows of the Jacobian i
- $\mathbf{J}_{oi}$ represents the last three rows of the Jacobian i
- $\mathbf{S}(\mathbf{p}_j^i)$ is a skew-symmetric matrix with input $\mathbf{p}_j^i$
- $\mathbf{I}$ is an identity matrix

The relative twist can be rewritten as:

$$\begin{bmatrix} \dot{\mathbf{p}}_R^L \\ \dot{\boldsymbol{\omega}}_R^L \end{bmatrix} = \begin{bmatrix} -\mathbf{R}_0^L \mathbf{J}_{pL} + \mathbf{S}(\mathbf{p}_R^L)\, \mathbf{R}_0^L \mathbf{J}_{oL} & \mathbf{R}_0^R \mathbf{J}_{pR} \\ \mathbf{R}_0^L \mathbf{J}_{oL} & \mathbf{R}_0^R \mathbf{J}_{oR} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_L \\ \dot{\mathbf{q}}_R \end{bmatrix} \tag{4.31}$$

$$\begin{bmatrix} \dot{\mathbf{p}}_R^L \\ \dot{\boldsymbol{\omega}}_R^L \end{bmatrix} = \left[\begin{bmatrix} \mathbf{I} & -\mathbf{S}(\mathbf{p}_R^L) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} -\mathbf{R}_0^L & \mathbf{0} \\ \mathbf{0} & -\mathbf{R}_0^L \end{bmatrix} \begin{bmatrix} \mathbf{J}_{pL} \\ \mathbf{J}_{oL} \end{bmatrix} \quad \begin{bmatrix} -\mathbf{R}_0^L & \mathbf{0} \\ \mathbf{0} & -\mathbf{R}_0^L \end{bmatrix} \begin{bmatrix} \mathbf{J}_{pR} \\ \mathbf{J}_{oR} \end{bmatrix} \right] \begin{bmatrix} \dot{\mathbf{q}}_L \\ \dot{\mathbf{q}}_R \end{bmatrix} \tag{4.32}$$

That allows us to write a more compact form of the relative jacobian, $\mathbf{J}_r$:

$$\mathbf{J}_r = \begin{bmatrix} -\boldsymbol{\psi}_R^L \boldsymbol{\omega}_0^L \mathbf{J}_L & \boldsymbol{\omega}_0^L \mathbf{J}_R \end{bmatrix} \tag{4.33}$$

where:

$$\boldsymbol{\psi}_R^L = \begin{bmatrix} \mathbf{I} & -\mathbf{S}(\mathbf{p}_R^L) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad \boldsymbol{\omega}_0^L = \begin{bmatrix} -\mathbf{R}_0^L & \mathbf{0} \\ \mathbf{0} & -\mathbf{R}_0^L \end{bmatrix} \tag{4.34}$$

The wrench transformation matrix $\boldsymbol{\psi}_R^L$ contribution in the relative linear velocity can be negligible when the rotational velocity of the left end-effector is close to zero. It is clearly seen that in order to derive the relative jacobian $\mathbf{J}_r$ using the proposed method, one only needs to derive the wrench transformation matrix $\boldsymbol{\psi}_R^L$ and the rotation matrix $\boldsymbol{\omega}_R^L$, then incorporate the Jacobians of the standalone manipulators $\mathbf{J}_L$ and $\mathbf{J}_R$ to form the relative Jacobian $\mathbf{J}_r$.

4.4.5 Payload manipulation constraints

The relative pose constraints are imposed to hold the payload along the task period. In addition to that constraint, in order to have a physical consistent manipulation, the manoeuvred object has to be subject to reasonable forces and moment to be moved in space. Pretending to see the payload as a box, Figure 4.5, it means that Newton and Euler equations (4.36) need to be satisfied and imposed as hard constraints at each multiple shooting node k:

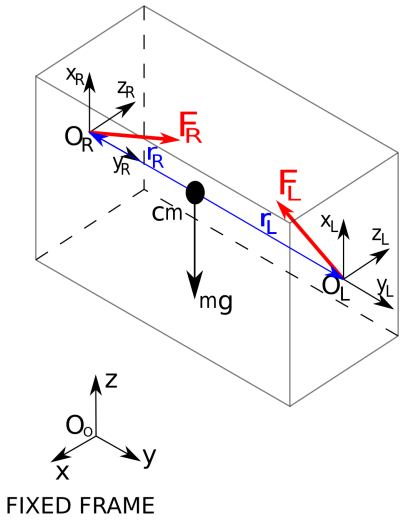


Figure 4.5: Payload free body diagram

$$\begin{cases} \mathbf{F}_L + \mathbf{F}_R - m\boldsymbol{g} = m\mathbf{a}_{cm} \\ \mathbf{r}_L \times \mathbf{F}_L + \mathbf{r}_R \times \mathbf{F}_R = \mathbf{I}\dot{\boldsymbol{\omega}}_{cm} \end{cases} \tag{4.35}$$

where m is the payload mass, $\mathbf{I}$ is the moment of inertia matrix, $\mathbf{a}_{cm}$ is the center of mass acceleration and $\dot{\boldsymbol{\omega}}_{cm}$ is the angular acceleration. Because of the quasi-static assumption mentioned in (4.4), those two equations are simplified, at each time step, as:

$$\begin{cases} \mathbf{F}_L + \mathbf{F}_R - m\boldsymbol{g} = 0 \\ \mathbf{r}_L \times \mathbf{F}_L + \mathbf{r}_R \times \mathbf{F}_R = 0 \end{cases} \tag{4.36}$$

4.4.6 Friction Cone

The contact forces must meet friction cone constraints in order to prevent payload/end-effector slippage. In order to keep the hand away from slipping on the payload, the tangential local forces must be smaller than the normal force times the static coefficient of friction. In this thesis, the lateral force in the tangential plane will be required to independently satisfy the condition. An additional requirement is that only positive normal force can be applied in order to grasp the object.

Friction cone theory - Coulomb friction

Referring to the Figure 4.6, we can state that a contact will remain fixed as long as the contact force $\mathbf{f}_C = m\mathbf{g} - \mathbf{F}_{ext}$ lies within the Coulomb friction cone $\mathbf{C}$. As soon as $\mathbf{f}_C$ exits $\mathbf{C}$, the contact switches to the sliding mode.

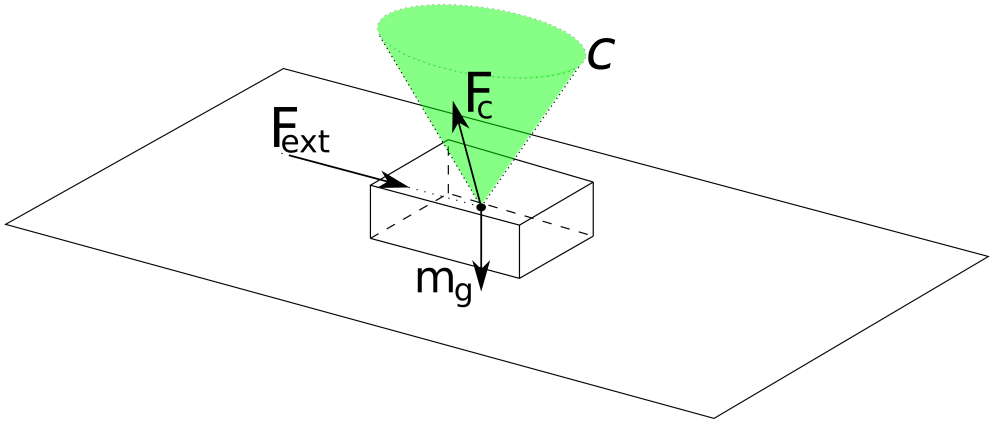


Figure 4.6: Theory of friction cone

The property $\mathbf{f}_C \in \mathbf{C}$ is called the contact-stability condition for the contact mode: as long as it is fulfilled, the contact remains fixed. Consider the set of points $\mathbf{C}_i$ where the robot contacts its environment. Once we introduce $\mathbf{f}_{Ci}$ as the contact force exerted at $\mathbf{C}_i$ by the environment onto the robot and the unit normal $\mathbf{n}_i$ at $\mathbf{C}_i$, pointing from the environment to the robot, we can define:

- the normal component $\mathbf{f}_i^n = (\mathbf{n}_i \cdot \mathbf{f}_{Ci})\mathbf{n}_i$.
- the tangential component $\mathbf{f}_i^t = \mathbf{f}_{Ci} - (\mathbf{n}_i \cdot \mathbf{f}_{Ci})\mathbf{n}_i$.

A contact point remains in the fixed contact mode while its contact force $\mathbf{f}_{Ci}$ lies inside the friction cone:

$$(\mathbf{f}_{Ci} \cdot \mathbf{n}_i) > 0 \text{ and } ||\mathbf{f}_i^t||^2 \leq \boldsymbol{\mu}_i(\mathbf{n}_i \cdot \mathbf{f}_{Ci})\mathbf{n}_i \tag{4.37}$$

where μ_i is the static friction coefficient at contact C_i. The Euclidean norm$|| \cdot ||^2$ in this definition represents friction cones with circular sections. Although more realistic, this model presents some computational challenges and a common practice is to consider its linear approximation, Figure 4.7. A contact point remains in the fixed contact mode while its contact force f_{Ci} lies inside the linearized friction. This approximation can be made

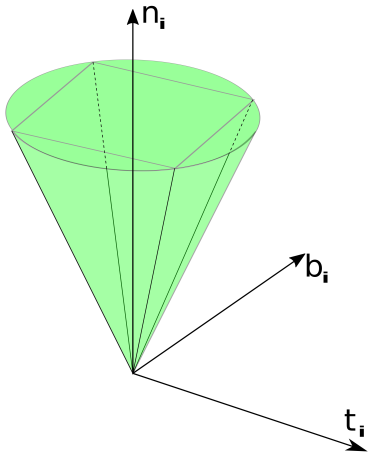


Figure 4.7: Linearized friction cone

as close as desired to the original one by increasing the number of edges n of the section polygon. For example, the four-sided friction pyramid in Figure 4.7 obtained can be written:

$$\begin{aligned} &\mathbf{f}_{Ci} \cdot \mathbf{n}_i > 0 \\ &|\mathbf{f}_{Ci} \cdot \boldsymbol{t}_i| \leq \mu_i \left(\mathbf{f}_{Ci} \cdot \boldsymbol{n}_i\right) \\ &|\boldsymbol{f}_{Ci} \cdot \boldsymbol{b}_i| \leq \mu_i \left(\boldsymbol{f}_{Ci} \cdot \boldsymbol{n}_i\right) \end{aligned} \tag{4.38}$$

with (t_i,b_i) any basis of the tangential contact plane such that (t_i,b_i,n_i) is a direct frame.

Friction cone constraints definition

Let's pretend the payload to be a box as it is shown in fig.(4.8). The manipulator applies forces at the contact points $\mathbf{O}_L$ and $\mathbf{O}_R$. In order to derive the matrix formulation of the friction cone constraint, we define the normal vectors $\mathbf{n}_L, \mathbf{n}_R$ on the box expressed in the local frames $(\mathbf{x}_L, \mathbf{y}_L, \mathbf{z}_L)$ and $(\mathbf{x}_R, \mathbf{y}_R, \mathbf{z}_R)$, pointing from the environment to the robot. Then we construct two frames, $(\mathbf{t}_L, \mathbf{b}_L, \mathbf{n}_L)$ and $(\mathbf{t}_R, b_R, \mathbf{n}_R)$, based on the normal vector defined previously. Then, we project such versors in the local frame $\{\mathbf{L}\}, \{\mathbf{R}\}$. At the contact point $\mathbf{O}_L$:

$$\mathbf{t}_L = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad b_L = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{n}_L = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{4.39}$$

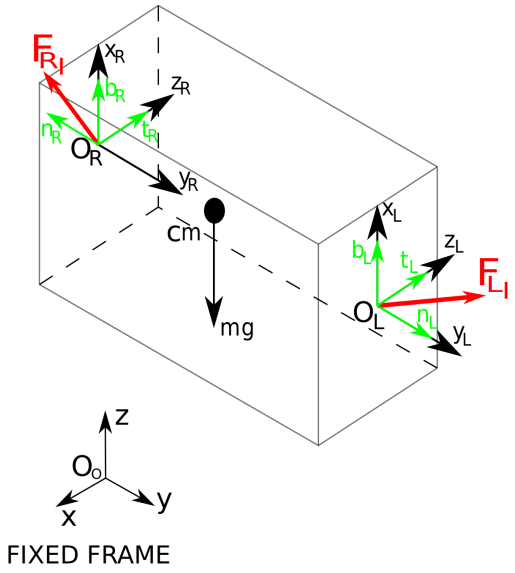


Figure 4.8: Payload friction cone

The force versors in the local frame at the second contact point $\mathbf{O}_R$:

$$\mathbf{t}_R = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad b_R = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{n}_R = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \tag{4.40}$$

We now consider $\mathbf{F}^C_{L_l}$, $\mathbf{F}^C_{R_l}$, which are the local contact force exerted in the contact point by the environment onto the robot. We need to express those force as function of the force optimization variable of the problem, which are force expressed in the global reference frame and defined as the force that the robot exert on the environment. Said that we need to change the sign to the global variable and move them in the contact point local frame:

$$\mathbf{F}^C_{L_l} = -\mathbf{R}^L_0 \mathbf{F}_L \tag{4.41}$$
$$\mathbf{F}^C_{R_l} = -\mathbf{R}^R_0 \mathbf{F}_R \tag{4.42}$$

Knowing that it is now possible to express the linearized constraints (4.38) in the form of (4.43):

$$\mathbf{O}_i \in [L,R] \begin{cases} \mathbf{F}_{i_l}^C \cdot \mathbf{n}_i > 0 \\ \left|\boldsymbol{F}_{i_l}^C \cdot \boldsymbol{t}_i\right| \leq \boldsymbol{\mu}\left(\mathbf{F}_{i_l}^C \cdot \boldsymbol{n}_i\right) \\ \left|\boldsymbol{F}_{i_l}^c \cdot \boldsymbol{b}_i\right| \leq \boldsymbol{\mu}\left(\boldsymbol{F}_{i_l}^C \cdot \boldsymbol{n}_i\right) \end{cases} \tag{4.43}$$

which can be rewritten in matrix form as follow, once the (4.41),(4.42) are substituted:

$$\text{At contact point } \mathbf{O}_L : \begin{bmatrix} 0 & -1 & 0 \\ 0 & -\mu & 1 \\ 0 & -\mu & -1 \\ 1 & -\mu & 0 \\ -1 & -\mu & 0 \end{bmatrix} \mathbf{F}_{L_l}^C = - \begin{bmatrix} 0 & -1 & 0 \\ 0 & -\mu & 1 \\ 0 & -\mu & -1 \\ 1 & -\mu & 0 \\ -1 & -\mu & 0 \end{bmatrix} \mathbf{R}_0^L \mathbf{F}_L = \mathbf{c}_L \mathbf{F}_L \tag{4.44}$$

$$\text{At contact point } \mathbf{O}_R : \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mu & 0 \\ -1 & \mu & 0 \\ 0 & \mu & 1 \\ 0 & \mu & -1 \end{bmatrix} \mathbf{F}_{R_L}^C = - \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mu & 0 \\ -1 & \mu & 0 \\ 0 & \mu & 1 \\ 0 & \mu & -1 \end{bmatrix} \mathbf{R}_0^R \mathbf{F}_R = \mathbf{c}_R \mathbf{F}_R \tag{4.45}$$

4.5 Resulting optimal control problem

This section resumes the concepts introduced so far and tries to obtain a complete formulation of the optimal control problem. In Sections 4.2, 4.3 we discussed the optimization variable vector $\mathbf{w} = [\mathbf{x}, \dot{\mathbf{q}}, \mathbf{F}_L, \mathbf{F}_R]$ and the cost function. In Section 4.4 instead, in order to obtain a physically consistent solution, all the constraints needed to hold and manipulate the payload were analyzed. The result of the optimal control problem in 4.46 will allow the dual arm manipulator to execute the manipulation task specified in the cost function at its best without exceed the motor temperature bound. In fact, thanks to the thermal model encapsulated in the OCP, the robot would be able to predict the temperature evolution and automatically adjust the manipulation task in order to protect the more stressed joints.

Equation (4.46) represents the discretized version of the continuous optimal control and it is obtained exploiting the multiple shooting transcribing method discussed in Chapter 2, discretizing the time horizon t_h in N multiple shooting nodes.

One may notice that the relative pose constraint is imposed fixing the relative end effector position and orientation error. This approach is chosen over the relative twist constraint. That is because in order to solve the optimal control problem as fast as possible, the simulated time horizon is discretized over a limited number of steps N. Having a large time step would introduce the drift phenomena in case of relative twist constraint.

$$
\begin{aligned}
\min_{\mathbf{q},\mathbf{T},\dot{\mathbf{q}},\mathbf{F}_L,\mathbf{F}_R} \quad & \sum_{k=0}^{N}\sum_{i=0}^{n} w_i \mathbf{Y}_i(\mathbf{q}_k,\dot{\mathbf{q}}_k,\mathbf{F}_k) + w_q({\dot{\mathbf{q}}_k}^T\dot{\mathbf{q}}_k) + w_F \sum_{i\in\{L,R\}} (\mathbf{F}_{i_k}^T\mathbf{F}_{i_k}) \\
\text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}_{ini} && \text{(Initial conditions)},\\
& \dot{\mathbf{q}}_0 = \dot{\mathbf{q}}_{ini} && \text{(Initial conditions)},\\
& \mathbf{q}_{lb} \le \mathbf{q}_k \le \mathbf{q}_{ub} && \text{(Joint limits)},\\
& \dot{\mathbf{q}}_{lb} \le \dot{\mathbf{q}}_k \le \dot{\mathbf{q}}_{ub} && \text{(Velocity limits)},\\
& \boldsymbol{\tau}_{lb} \le \mathbf{C}(\mathbf{q}_k,\dot{\mathbf{q}}_k)\dot{\mathbf{q}}_k + \mathbf{g}(\mathbf{q}_k) + \mathbf{J}^T\mathbf{W}_k \le \boldsymbol{\tau}_{ub} && \text{(Torque limits)},\\
& \mathbf{F}_{L_k} + \mathbf{F}_{R_k} - m\tilde{\boldsymbol{g}} = 0 && \text{(Newton equations)},\\
& \mathbf{r}_{L_k}\times\mathbf{F}_{L_k} + \mathbf{r}_{R_k}\times\mathbf{F}_{R_k} = 0 && \text{(Euler equations)},\\
& \mathbf{p}^R_{Lk+1} - \mathbf{p}^R_{Lk} = 0 && \text{(Relative position constraint)},\\
& \mathbf{e}_{ok} - \mathbf{e}_{o0} = 0 && \text{(Relative angle constraint)},\\
& \mathbf{c}_i\mathbf{F}_{i_k} \le 0 \quad i\in[L,R] && \text{(Friction cone constraint)},\\
& \mathbf{q}_{k+1} = \mathbf{q}_k + \dot{\mathbf{q}}_k\Delta t && \text{(Continuity conditions)},\\
& \mathbf{T}_{k+1} = \mathbf{f}_{int}(\mathbf{q}_k,\dot{\mathbf{q}}_k,\mathbf{T}_k) && \text{(Continuity conditions)},\\
& \mathbf{T}_{k+1} \le \mathbf{T}_{bound} && \text{(Temperature constraint)}
\end{aligned}
\tag{4.46}
$$

4.6 Nonlinear model predictive control formulation

The optimal control problem in (4.46) generates the optimal open loop controls u(t) to send to the robot plant. That control result in the payload manipulation specified in the cost function and protects the joint from thermal failure. This optimal control problem is a block of a bigger controller, which is the model predictive control. The model predictive control was previously introduced as a method of control that utilizes a built-in descriptive model of a system to predict its behavior. These predictions are used to refine the optimization, anticipating future system responses. This logic is exploited to set-up a controller able to monitor the robot fatigue while executing a heavy task manipulation. Starting from Figure 4.9, we discuss how the MPC is implemented based on the optimal control problem obtained in the aforementioned section. The main idea is to optimize for the time horizon $[t_j, t_{j+1}]$ while executing the previously optimized trajectory $\mathbf{q}_j, \dot{\mathbf{q}}_j$.

The reader may immediately notice that the problem is split in two parts, the initialization and the run-time part.

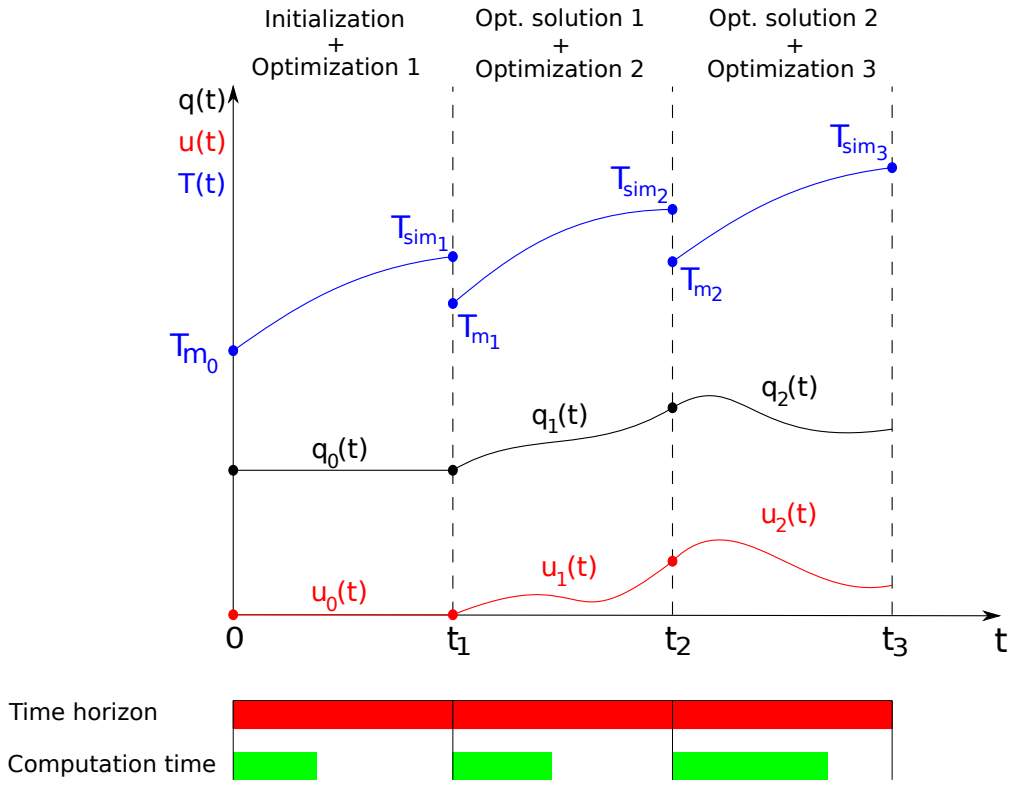


Figure 4.9: From OCP to NMPC scheme

MPC initialization

At the beginning of the MPC we need to initialize the optimization loop. As it is possible to spot in figure 4.9, in the first time window [0 - t_1] the robot is fed with an initial trajectory that describes the manipulation task. The joint temperature evolution in blue is predicted thanks to the thermal model specified in section 4.4. The temperature evolution prediction starts from the temperature measurement T_{m0} and its result is used as initial condition in the successive time horizon optimization. In fact, while the robot executes the trajectory $q_0(t)$, the optimization for the next time windows starts. Such optimization uses as initial condition the state vector:

$$x_1(t_1) = \begin{bmatrix} q_1(t_1) \\ T_1(t_1) \end{bmatrix} = \begin{bmatrix} q_0(t_1) \\ f_{int}(q_0(t), u_0(t), T_{m_0}) \end{bmatrix} \tag{4.47}$$

where $f_{int}(\cdot)$ is thermal model (4.20) that simulate the new initial temperature state at t_1 starting from the new measurement and the previously optimized trajectory. Once the first time horizon $[0, t_1]$ trajectory has been completely executed, the plant is fed with the new optimized trajectory obtained by the previous optimization. As soon as it starts, a new joint temperature measurement is taken and a new optimization problem for the successive time horizon is solved.

Run-time optimization

Once the j optimization has been solved and the *j-1* trajectory has been completely executed, we start in parallel the following processes:

- Measure the joint temperature T_{mj} from sensors.
- Send the j optimization problem output to the plant.
- Start the *j+1* optimization problem based on the initial state:

$$x_{j+1}(t_{j+1}) = \begin{bmatrix} q_{j+1}(t_{j+1}) \\ T_{j+1}(t_{j+1}) \end{bmatrix} = \begin{bmatrix} q_j(t_{j+1}) \\ f_{int}(q_j(t), u_j(t), T_{m_j}) \end{bmatrix} \tag{4.48}$$

Computation time

In Figure 4.9, one may notice the main computation time constraint that must be respected. The aforementioned approach is suitable only if each optimal control problem computation time is smaller than the time horizon we optimized for. In order to achieve such result, we may take advantage of some techniques.

Warm starting

The warmstarting is a technique that can speed up the optimal control problem solution and it consists in supplying an initial guess to the nonlinear problem solver. The first reason why it helps is that the solver has a vague idea about where the solution to a non-convex problem is, and want to start the search in that region. The second reason is that the problem involves singularities and generally tricky regions, and we want to ensure that the solver does not try to start there.

Reduce constraint tolerances

In order to speed up the solver one may think to reduce of some orders of magnitude the constraint tolerance. It turned out to be very effective and to result in a drastical reduction of the solver computation time.

Parameters dependency

The performance of many hard non-convex problem solvers strongly depends on their parameter settings. In our case, the main parameters that influence the solver are the multiple shooting node, the cost function weights, the simulated time horizon. At each set-up we need to find empirically the best value that allows the real-time optimization.

Chapter 5

Validation

This Chapter provides details of the proposed NMPC implementation and summarizes the simulation and experimental results. Section 5.1 and Section 5.2 give a description of the CENTAURO robot platform, and the exploited software framework which relies on CasADi, Pinocchio and the robotic operating system (ROS). Section 5.3 shows the NMPC controller implementation in the ROS environment whose results are finally discussed in Section 5.4 with the robot performing two heavy manipulation tasks.

5.1 The CENTAURO platform

The proposed NMPC approach is evaluated on the CENTAURO robot platform, which is the result of a four years project involving, among other partners, the Italian Institute of Technology (IIT). The robot, presented in Figure 5.1, has been designed with the purpose of accommodating the needs of real-world applications with high payload and harsh physical interaction demands like in disaster-response scenarios. It combines powerful manipulation capabilities with a more reliable quadrupedal hybrid wheeled-legged locomotion concept. The upper body shape was suggested by the need of performing bi-manual tasks as humans do, thus the upper body dimensions are set to approximately resemble the average human size. The locomotion profile of the robot, on the other hand, relies upon a quadrupedal lower-body to enable higher balancing capacity to deal with a wide variety of terrain conditions.

Figure 5.1: Centauro robot

5.1.1 Hardware description

The two arms show a quasi-anthropomorphic kinematics, see Figure 5.2. Each arm has 7-DoFs that enable the robot to manipulate the environment with large dexterity, providing also one degree of redundancy to overcome constraints that the arm may be subject to. It incorporates a 3-DoFs shoulder complex, an elbow joint and a 3-DoFs wrist module. The shoulder follows a typical pitch-yaw-roll structure, whereas the wrist employs a yaw-pitch-yaw arrangement.

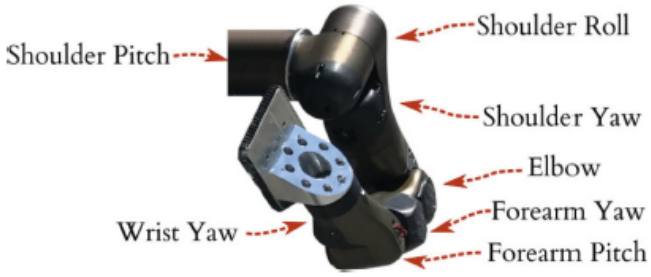


Figure 5.2: Centauro robot arm

The CENTAURO legs, shown in Figure 5.3, incorporate 5-DoFs each, with three upper joints organized in a yaw–pitch–pitch configuration and two lower joints for wheel orientation and steering. Since the spider-like case demonstrates benefits of reduced joint effort, the first 3 DoFs are arranged using a spider-like hip configuration. That, in fact, can provide better stability when executing manipulations with high payloads and interaction forces.

The robot pelvis accommodates the hip yaw actuator of each leg, and houses the robot battery, power distribution electronics, and a computation unit arranged with high-performance GPU responsible for system high level control and motion planning. The robot torso is mounted to the pelvis through a yaw joint permitting the rotation of the upper-body in the transverse plane.

5.1.2 Software description

The XBotCore framework [22] has been employed as a control framework. XBotCore is an open-source, real-time, platform independent software which does not only function as the robot software middleware, but it also handles the robot real-time (RT) control schemes. This software makes it possible to seamlessly program and control any robotic platform while providing a canonical abstraction that hides the specifics of the underlying hardware. Cartesian space motion is performed by the CartesI/O motion library [23], which can consider multiple Cartesian tasks and constraints under hard/soft priorities.

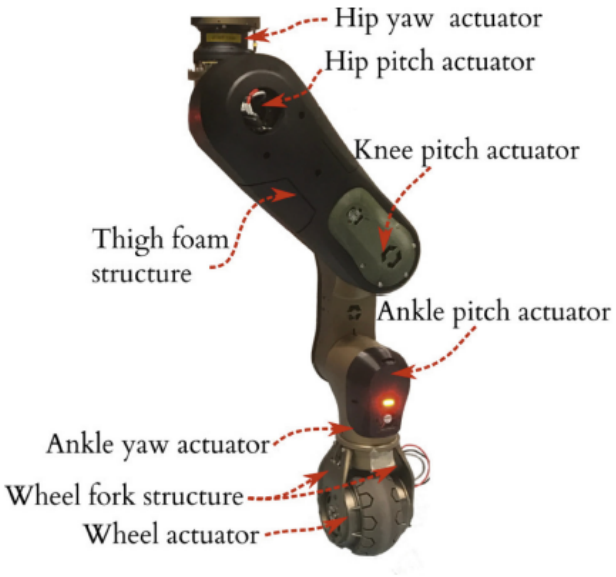


Figure 5.3: Centauro robot leg

5.2 Tools overview

The NMPC algorithm introduced in Chapter 4 is implemented using several softwares. For instance, the robot dynamics is computed thanks to the Pinocchio library whereas the optimal control problem is transcribed to a nonlinear problem and then solved using the CasADi framework. The whole algorithm, instead, is developed in the ROS environment. The next paragraphs will briefly introduce these tools.

5.2.1 CasADi

CasADi [24] is a free, open-source, general purpose software tool for nonlinear optimization and analytical differentiation. It is best described as a minimalistic computer algebra system (CAS) implementing automatic differentiation (AD)[1]. It is mainly used to model and solve optimization problems in a flexible, interactive and numerically efficient way. It is written in C++ and thanks to a full-featured interface it is very easy to use in Python programming language. Thanks to its simplicity, it has been used successfully in multiple fields ranging from process control to robotics and aerospace. It facilitates rapid and efficient implementation of different methods for numerical optimal control and for NMPC. CasADi treats optimal control problems using a different approach with respect to other existing tools because rather than providing the end user with a black box OCP solver, it

[1]Automatic differentiation (AD), is a technique for evaluating derivatives of computer represented functions, which has proved useful in nonlinear optimization.

provides a framework that allows advanced users to implement their method of choice, with any complexity. One of its main advantages is that everything is a matrix and all matrices are sparse and stored in the compressed column format (CSS) which helps to save a significant amount of memory and speed up the processing of that data. This work exploits CasADi to transcribe the fatigue aware heavy manipulation optimal control problem into a nonlinear problem writing a multiple shooting method and then solving it using an interior point nonlinear solver.

5.2.2 The Pinocchio library

Pinocchio [25] is an open-source software framework that implements fast and flexible rigid body dynamics algorithms and their analytical derivatives. It has been written in C++ for efficiency reasons and uses the Eigen library [2] for linear algebra routines. Pinocchio does not only include standard algorithms employed in robotics but it also provides additional features essential for control, planning and simulation of a robotic system. In this works Pinocchio was used to compute the symbolic formulation of the forward kinematic and the inverse dynamics given the robot *Unified robot description format* (URDF). The URDF file describes the parts that compose the robot and the way they are connected through joints.

5.2.3 ROS - Robotic Operating System

ROS (Robot Operating System) is an open-source Linux-based reusable robotic middleware that allows to program robotic applications quickly and easily. Figure 5.4 shows the standard architecture of the ROS software system, which consists of libraries, packages and software modules encapsulated as nodes, including the master node and functional nodes. The master node administrates and monitors the running of the functional nodes and their peer-to-peer communications. Each node represents a single running process and together they are at the center of ROS programming since they take actions based on information received from other nodes, sends information to other nodes, or sends and receives requests for actions to and from other nodes. Topics are buses over which nodes send and receive messages. To send messages to a topic, a node must publish to said topic, while to receive messages it must subscribe. The types of messages passed on a topic vary widely and can be user-defined. The content of these messages can be sensor data, motor control commands, state information, actuator commands, or anything else. A node may also advertise services. A service represents an action that a node can take which will have a single result. ROS's core functionality is augmented by a variety of tools that allow developers to visualize and record data, easily navigate the ROS package structures, and create scripts automating complex configuration and setup processes. The addition of these tools greatly increases the capabilities of systems using ROS by simplifying and providing solutions to a number of

[2]Eigen is a high-level open-source C++ library for linear algebra, matrix and vector operations and related algorithms.

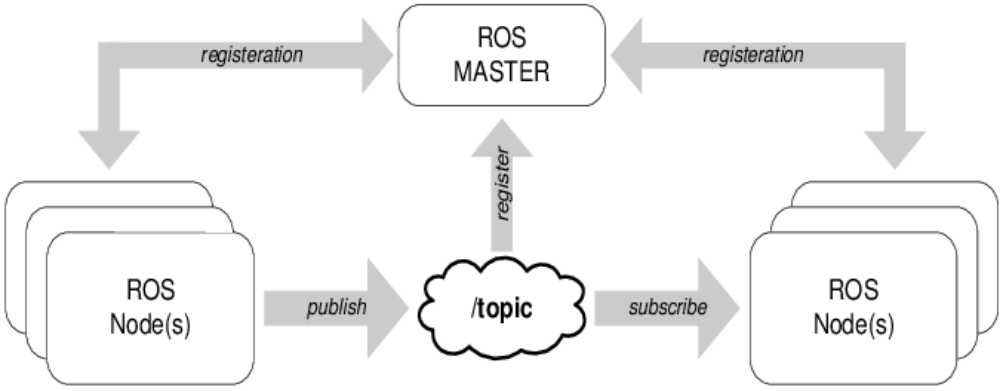


Figure 5.4: Robotic operating architecture

common robotics development. These tools are provided in packages like any other algorithm, but rather than providing implementations of hardware drivers or algorithms for various robotic tasks, these packages provide task and robot-agnostic tools which come with the core of most modern ROS installations.

5.3 Model predictive control in ROS

Chapter 4 introduced the model predictive controller logic based on the optimal control problem, whose solution describes fatigue aware heavy manipulation task over a fixed future time horizon. Recall that the NMPC simultaneously solves the j^{th} optimal control while sending the $j\text{-}1^{th}$ optimized trajectory to the real robot. This thesis exploits ROS to develop an application that performs both tasks together. More precisely the nonlinear model predictive controller execution is based on two ROS nodes, the *Optimal control node* and the *Interpolator node*, see Figure 5.5. The first one computes the future optimal controls while the second sends in real-time the trajectory optimized at the previous step.

Optimal control node

It solves the optimal control problem with prediction horizon t_h transcribing the optimal control to a nonlinear problem according to the direct multiple shooting method. Then, the NLP is passed to the IPOPT solver that finds the optimal solution. The computation time required to solve the optimal control problem has to be shorter than the prediction horizon it optimizes for, $t_c < t_h$. To respect this constraint, the node takes advantage of the slow motor thermal dynamics optimizing over a large prediction horizon using a limited number of multiple shooting nodes. Once the optimal solution is found it waits that the interpolator node has sent all the previous trajectory to the robot plant and then it publishes the new trajectory to the interpolator node, takes a new motor temperature measurement and starts to solve a new optimal control problem.

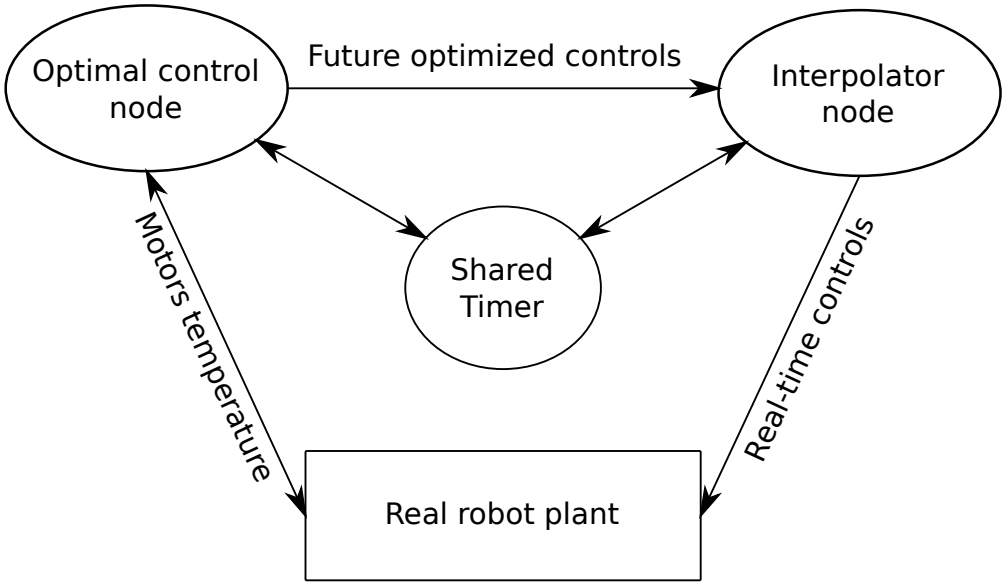


Figure 5.5: NMPC implementation is ROS

Interpolator node

Since the optimal control node discretizes the OCP using a limited number of multiple shooting nodes, the resulting optimized trajectory and control law are vectors of limited dimension as well. The interpolator node, which is a subscriber of the optimal control node, stores the trajectory received by the first node and sends its interpolated version to the real robot plant with a certain frequency. The interpolation is done linearly and it is performed exploiting the encapsulated ROS timer.

5.4 Experimental set-up

The proposed NMPC algorithm has been validated in two bimanual manipulation tasks. In the first task, the robot is asked to hold a payload in a user-defined Cartesian space position, whereas in the second test the robot holds a box and moves its center of mass over a predefined circular trajectory. The position tasks are expressed in the world reference frame, which is identified on the ground in the robot URDF, Figure 5.6. As it will be shown in the following, the proposed NMPC will allow the robot to execute the tasks assigned while adapting the robot configuration at run-time in order to prevent motor overheating.

5.4.1 Thermal model assumptions

The thermal model introduced in Chapter 4 was used to impose the thermal constraint in the optimal control problem. Its dynamics depends on unknown parameters that need to be properly identified. The next simulations assume values that are not the real one and leaves

as future work the thermal model parameter identification. Because of that assumption, the NMPC logic explained in Chapter 4 needs to be slightly modified. In fact, to solve the *j+1* optimization problem, the initial temperature conditions are not obtained by integration of the measured temperature, but using the final temperature obtained by the previous optimization problem:

$$x_{j+1}(t_{j+1}) = \begin{bmatrix} q_{j+1}(t_{j+1}) \\ T_{j+1}(t_{j+1}) \end{bmatrix} = \begin{bmatrix} q_j(t_{j+1}) \\ T_j(t_{j+1}) \end{bmatrix} \tag{5.1}$$

5.5 Payload holding experiments

The parameters used in the payload holding test are resumed in Appendix. B.1, whereas the CENTAURO platform limits such as joint angle and joint velocities limits are illustrated in Appendix A. The payload holding experiment, Figure 5.6, is done exploiting three different approaches:

- NMPC approach without temperature constraint.
- NMPC approach with temperature constraint.
- Minimum effort approach.

The three solutions are compared in order to show the benefits that the proposed approach can bring to the thermal fatigue problem in the heavy manipulation task. The payload holding task is assigned to the robot by defining the cost function in a proper way. The NMPC cost function is defined as follows:

$$\mathbf{J} = \sum_{k=0}^{N} w_p ||\mathbf{e}_{pk}|| + w_q(\dot{\mathbf{q}}_k{}^T \dot{\mathbf{q}}_k) + w_f(\mathbf{F}_k^T \mathbf{F}_k) \tag{5.2}$$

where $\mathbf{e}_{p_k}$ is the box position error with respect to the user defined holding position. The box position is computed as the mean value of the two end effectors positions. The cost function also considers regularization terms on the global contact forces and joint velocities. The cost function for a minimum effort problem instead, is written similarly but it also minimizes torques:

$$\mathbf{J} = \sum_{k=0}^{N} w_p ||\mathbf{e}_{pk}|| + w_q(\dot{\mathbf{q}}_k{}^T \dot{\mathbf{q}}_k) + w_t(\boldsymbol{\tau}_k^T \boldsymbol{\tau}_k) \tag{5.3}$$

5.5.1 Results discussion

The NMPC ROS application is executed three times to test the three aforementioned approaches. In all three cases, the algorithm starts with the initialization phase. While the

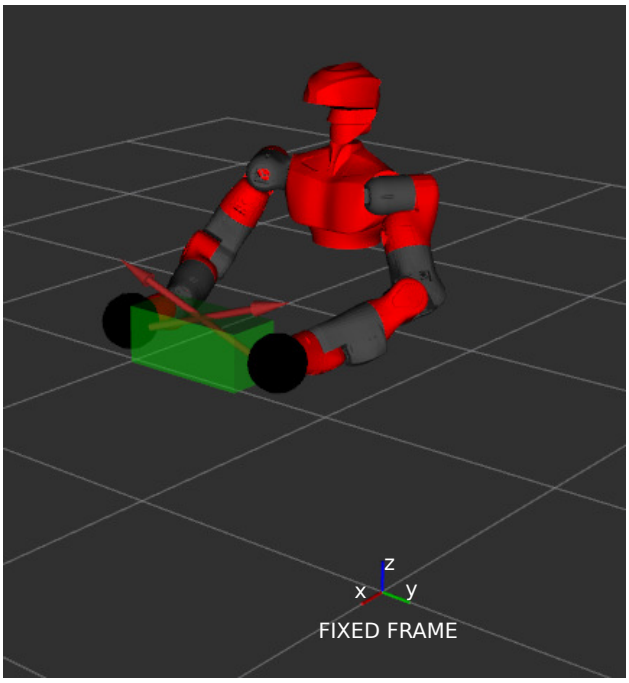


Figure 5.6: Payload holding task

optimal control node starts to solve the optimal control problem for the next prediction horizon, the real robot platform is fed by the interpolated node with the constant pose, received from the first node, that describes the box holding. The robot applies forces at the contact points that allow the payload holding and that respects the friction cone constraint. These contact forces require constant joint torque, whose value can be computed thanks to (4.14). This results in the robot configurations depicted in Figure 5.14a, 5.14b, 5.14c whereas the temperature evolution can be integrated with the thermal model (4.17) and used as initial values for the next optimization, Figure 5.7.

Once the initialization phase is over, the *optimal control node* sends the new t_h optimized control law to the *interpolator node* and immediately starts solving the successive optimization problem. The interpolator node feeds the real plant with the new trajectory, which now differ among the three problems. From Figure 5.12 and Figure 5.14d, it may be seen that without temperature constraint the robot would continue holding the box at the position required by the task. Doing so, as shown in Figure 5.11, it would violate the temperature constraint on the left and right shoulder pitch and elbows. The NMPC solution that takes into account the temperature constraint is expected to prevent that by adjusting the robot configuration. Figures 5.12, 5.14e shows the results. The new optimized trajectory reconfigures the robot posture to bound the temperature on the too stressed joints, continuing the task execution. This result is also observable in Figure 5.8 and Figure 5.10, which depicts the robot joint angles and torque values evolution. Thanks to the new robot configuration,

torques are reduced as well.

Figure 5.12 depicts the box centre of mass position evolution. The reader may notice how the NMPC without thermal bounds executes the holding task perfectly since it has no reason to move the box away from its initial position, whereas the problem with temperature constraint moves at each iteration the box away from the desired reference to prevent motor thermal burnout. In the same figures, it is also shown the comparison to a minimum effort problem. Its result is completely different with respect to the presented NMPC approach since the task is drastically deteriorated independently of the joint temperatures. The minimum effort problems, in fact, aims at the torque reduction considering joints equally without taking their thermal fatigue into account.

Moreover, another interesting result can be observed in Figure 5.12. The reader may note that the robot prevents the thermal bound and tries to perform the task at its best at each iteration of the NMPC. In fact, in the time interval [20 - 40] s the robot drastically moves the box away from the initial position in order not to violate the thermal bound and then, once it has found a new configuration, it moves the box towards the initial position, minimizing the error position. The same behavior may be noticed in the time interval [60-80], it finds a new configuration that bound the temperatures on the elbows and minimizes the position error.

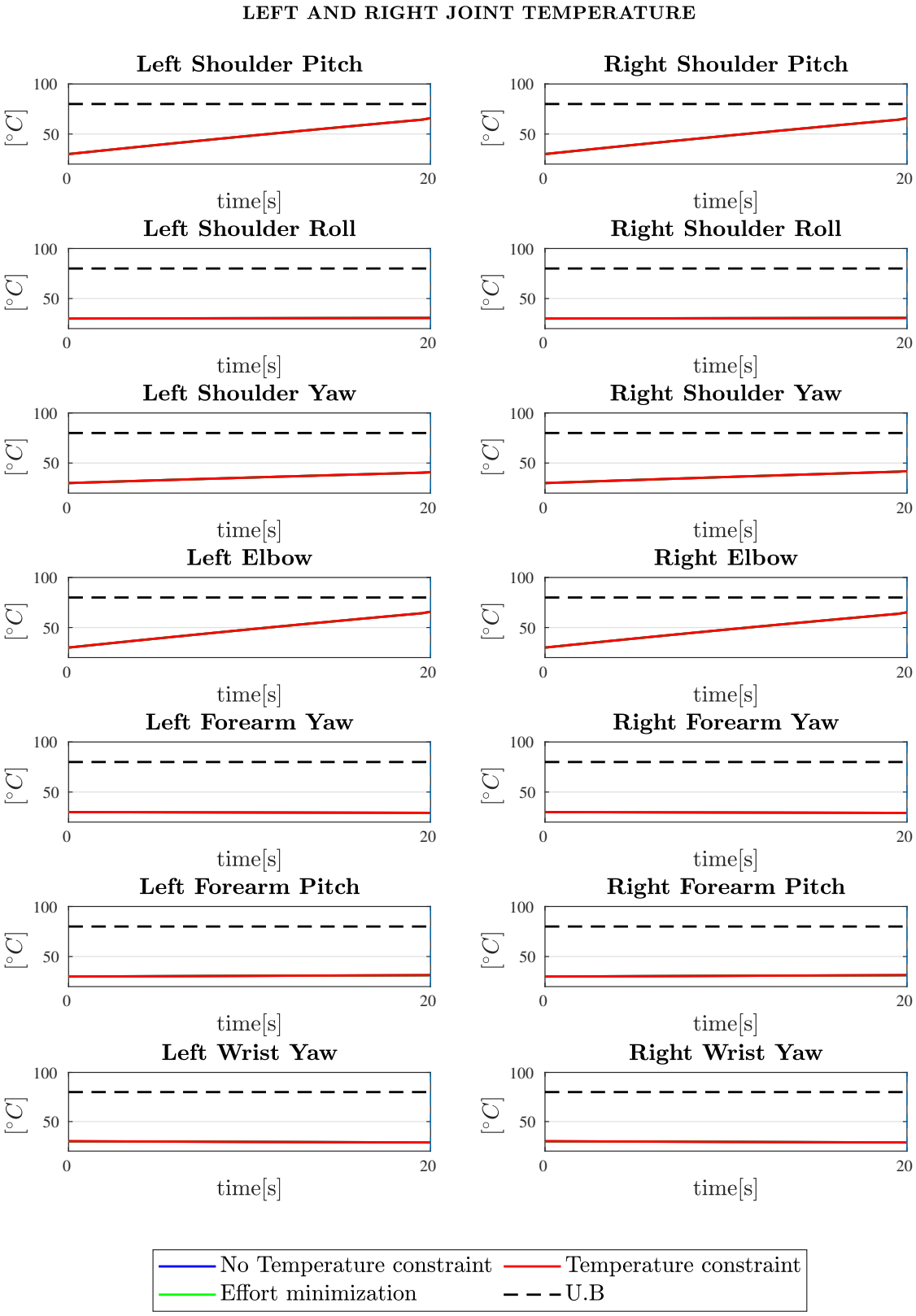


Figure 5.7: Payload holding - Joint temperatures evolution at initialization

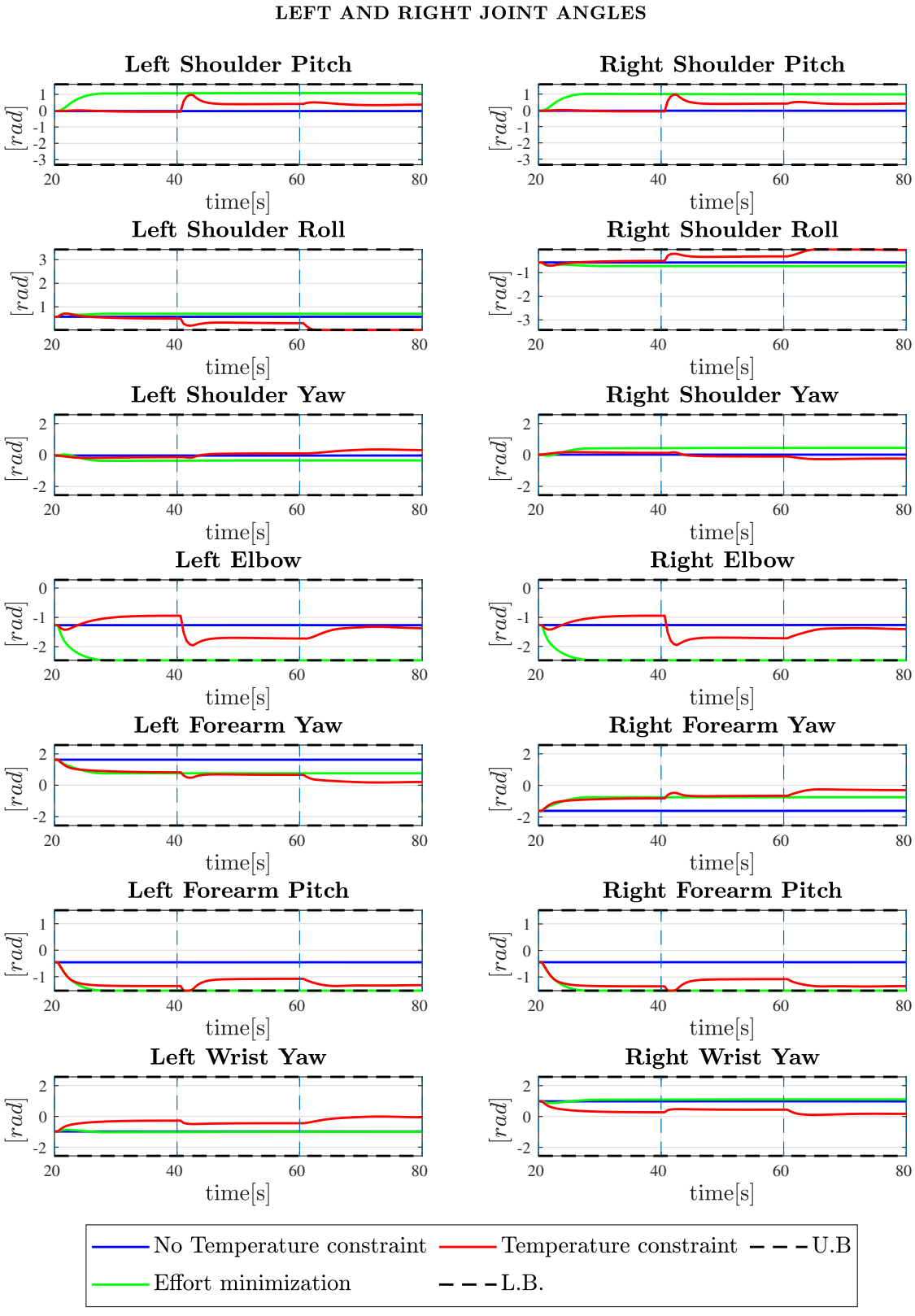


Figure 5.8: Payload holding - Run-time joint angles comparison

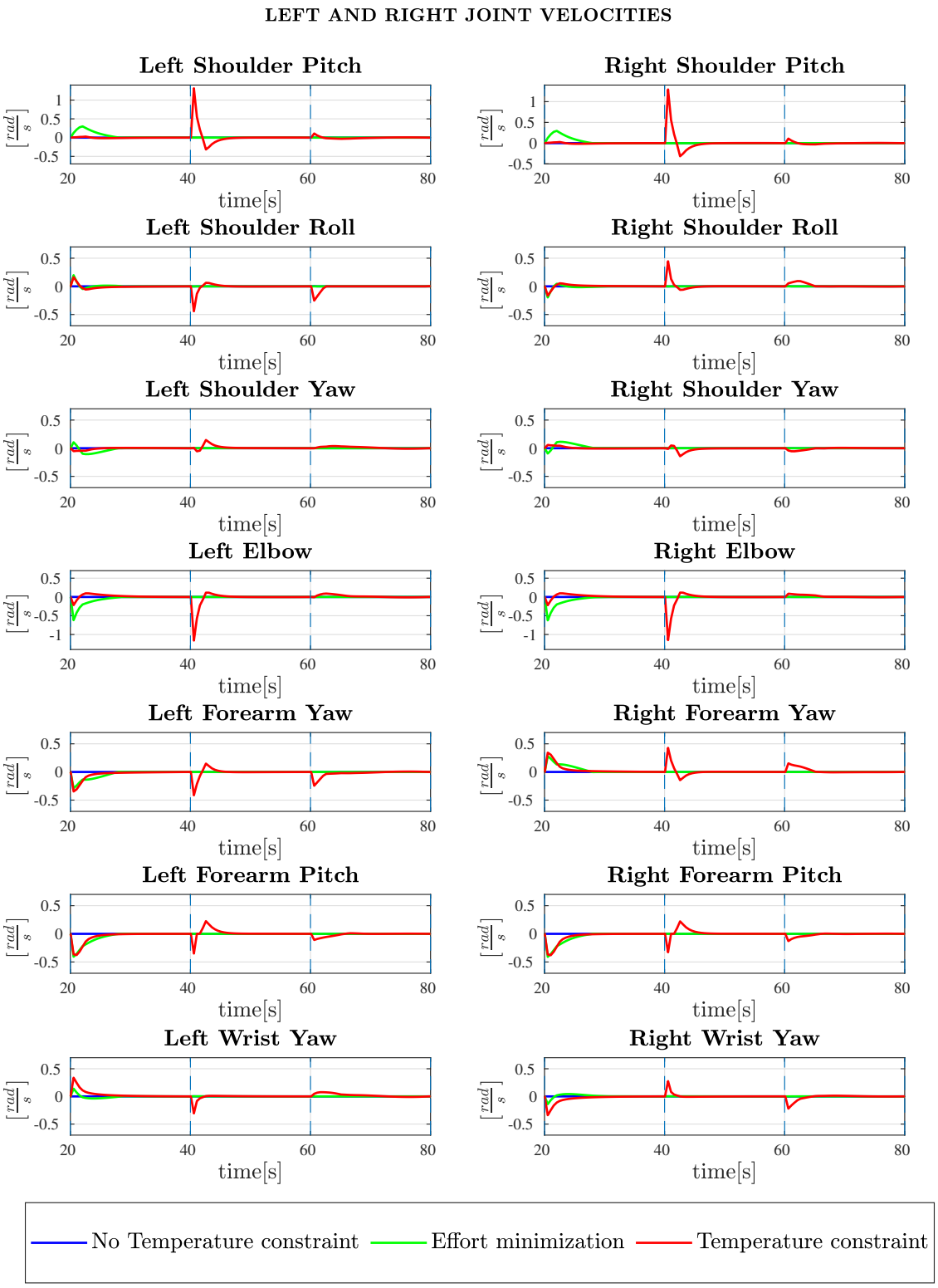


Figure 5.9: Payload holding - Run-time joint velocities comparison

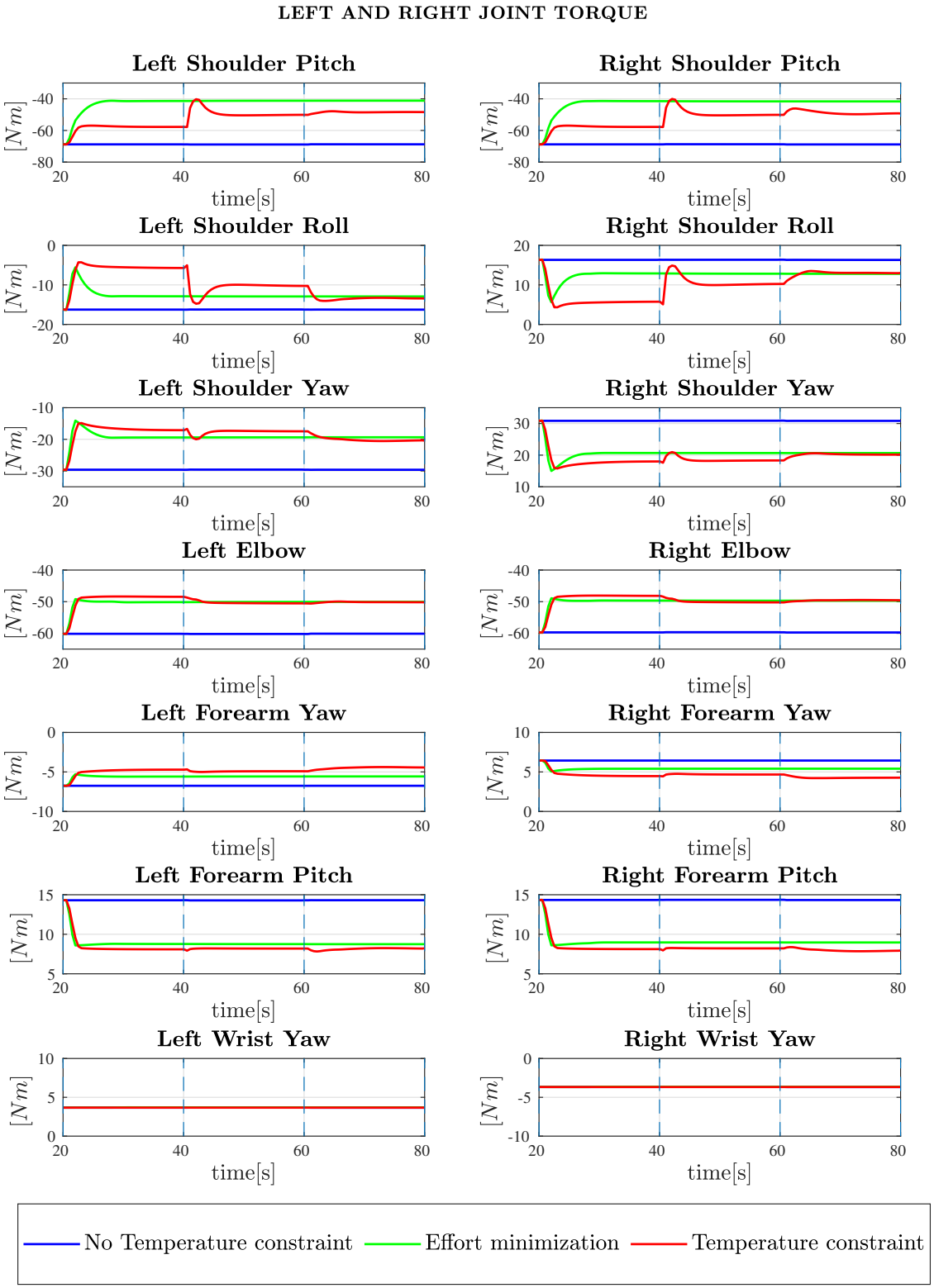


Figure 5.10: Payload holding - Run-time joint torques comparison

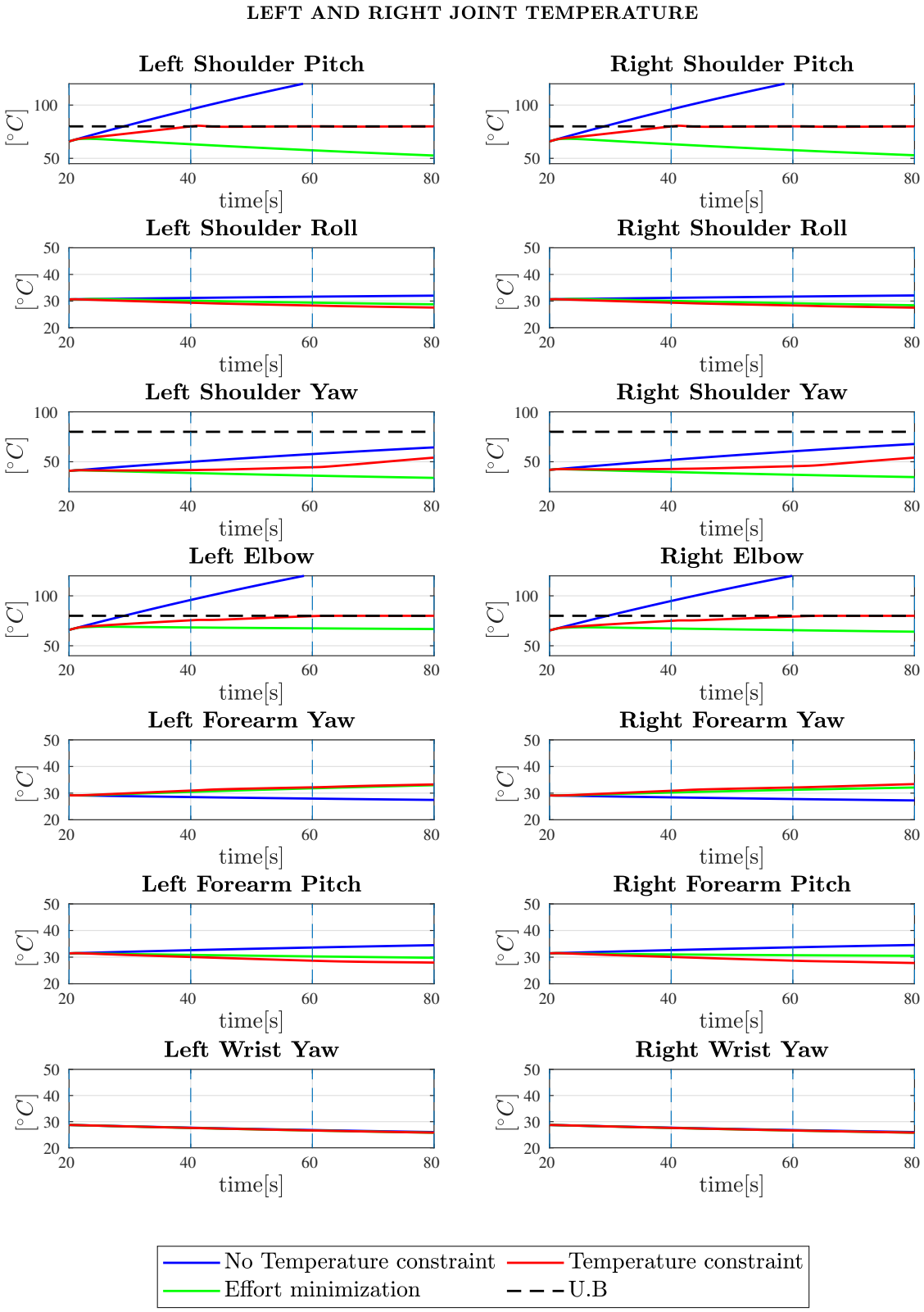


Figure 5.11: Payload holding - Run-time joint temperatures comparison

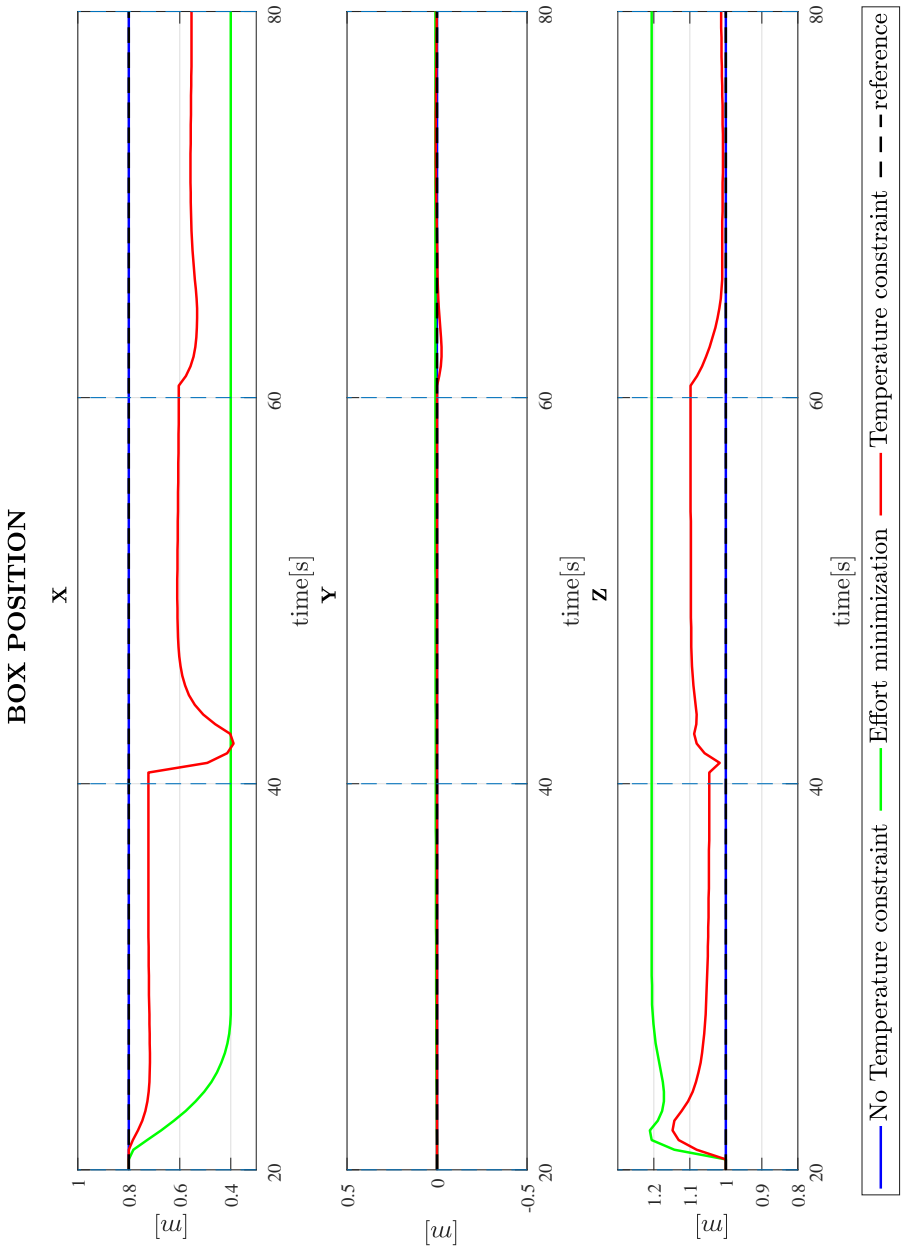


Figure 5.12: Payload holding - Run-time box position comparison

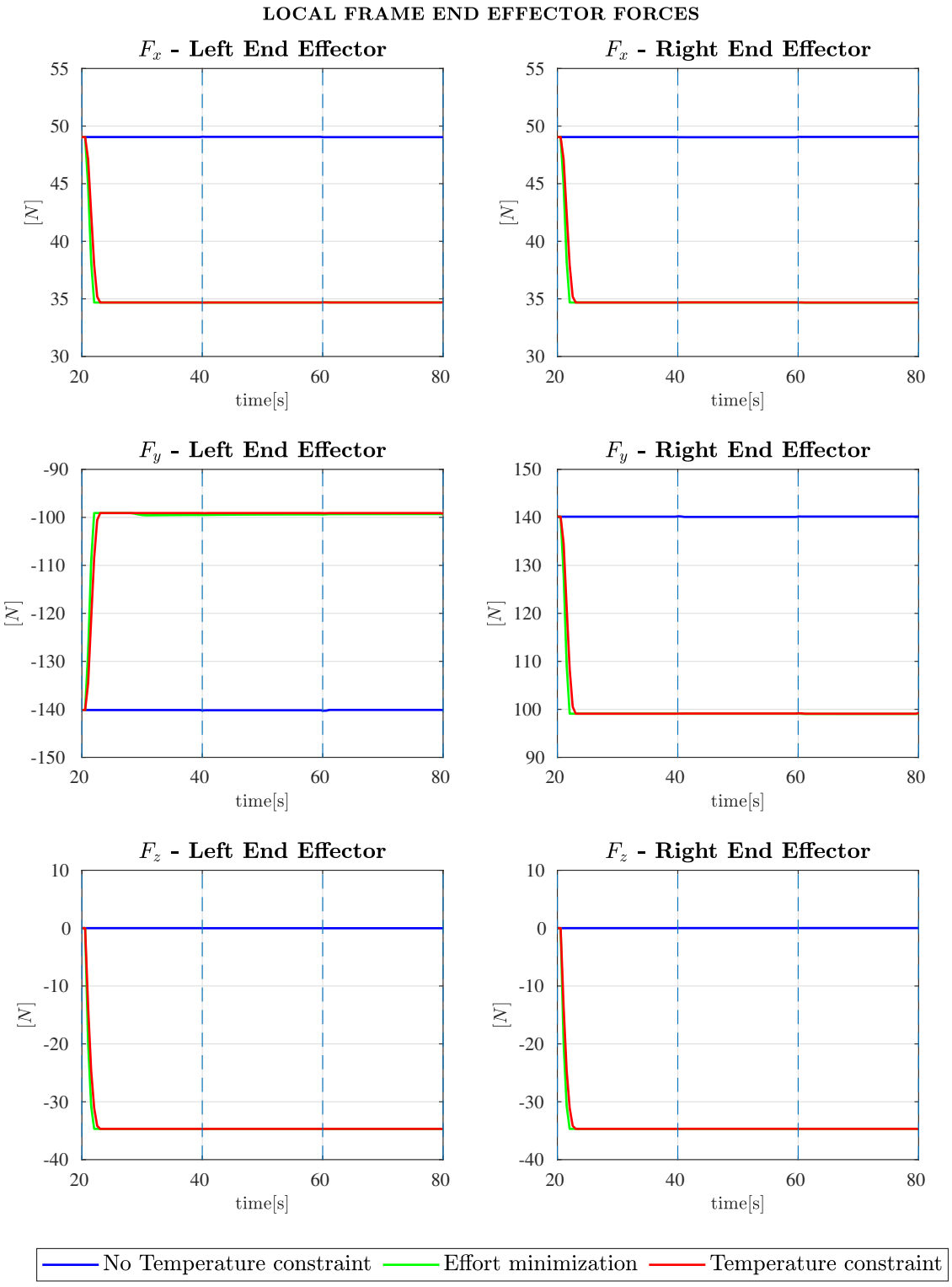
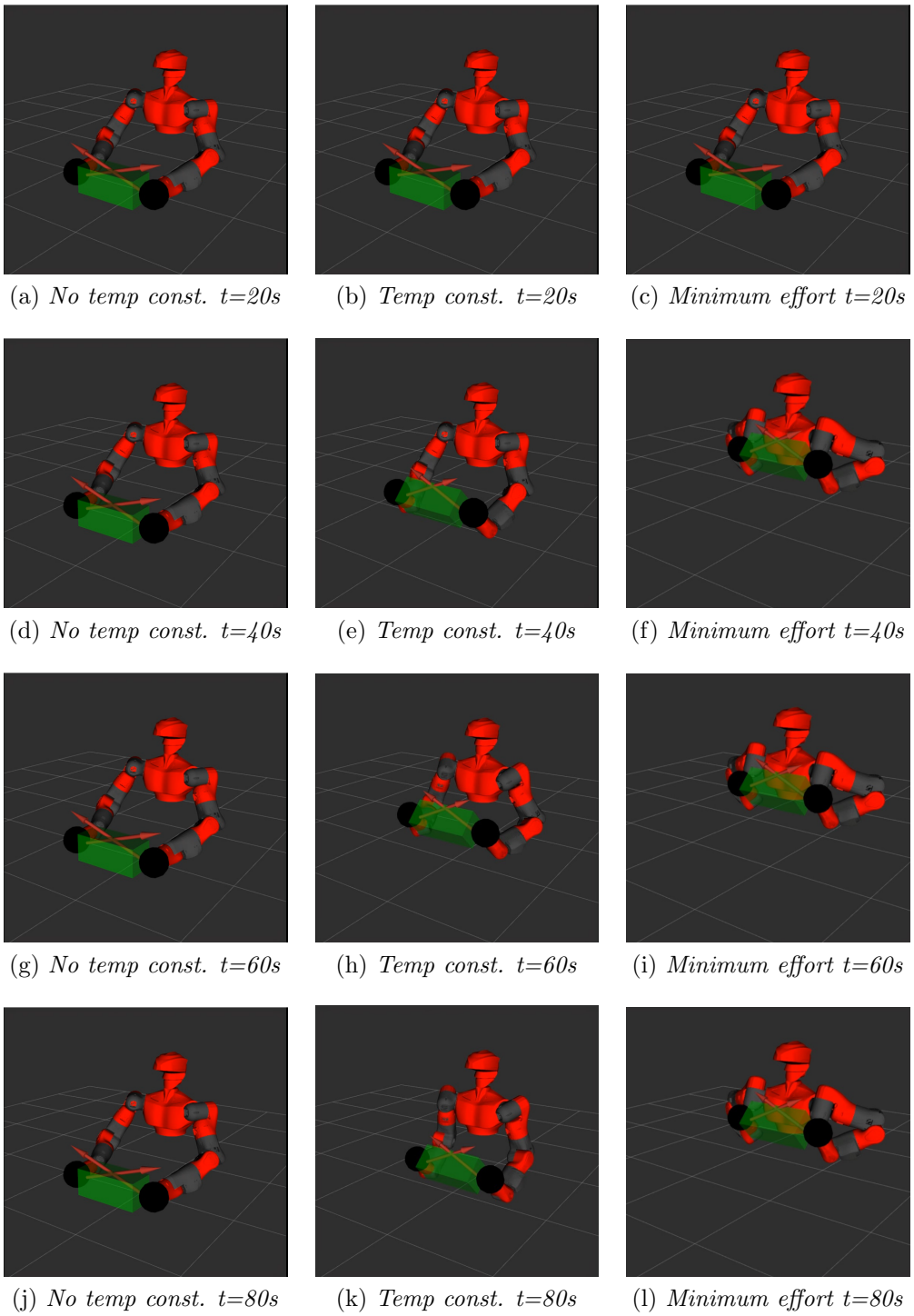


Figure 5.13: Payload holding - Run-time local contact forces comparison



(a) *No temp const. t=20s* (b) *Temp const. t=20s* (c) *Minimum effort t=20s*

(d) *No temp const. t=40s* (e) *Temp const. t=40s* (f) *Minimum effort t=40s*

(g) *No temp const. t=60s* (h) *Temp const. t=60s* (i) *Minimum effort t=60s*

(j) *No temp const. t=80s* (k) *Temp const. t=80s* (l) *Minimum effort t=80s*

Figure 5.14: Payload holding - Configurations evolution

5.6 Payload circular trajectory experiment

The parameters used in the payload holding test are resumed in Appendix. B.2, whereas the CENTAURO platform limits such as joint angle and joint velocities limits are illustrated in Appendix A. The NMPC solution is compared to the solution of the same problem without thermal constraints. The task is assigned to the robot by defining the cost function as follows:

$$\mathbf{J} = \sum_{k=0}^{N} w_p||\mathbf{e}_{pk}|| + w_o||\mathbf{e}_{ok}|| + w_q(\dot{\mathbf{q}}_k{}^T\dot{\mathbf{q}}_k) + w_f(\mathbf{F}_k^T\mathbf{F}_k) \tag{5.4}$$

where $\mathbf{e}_{pk}$ is the box position error with respect to the reference position $\mathbf{p}_{ref}$ at the k^{th} multiple shooting node. The reference position changes at each multiple shooting nodes and it is computed, starting from the box initial position $\mathbf{p}_{ini} = [x_{ini}\ y_{ini}\ z_{ini}]^T$, by discretizing the circular path on N points.

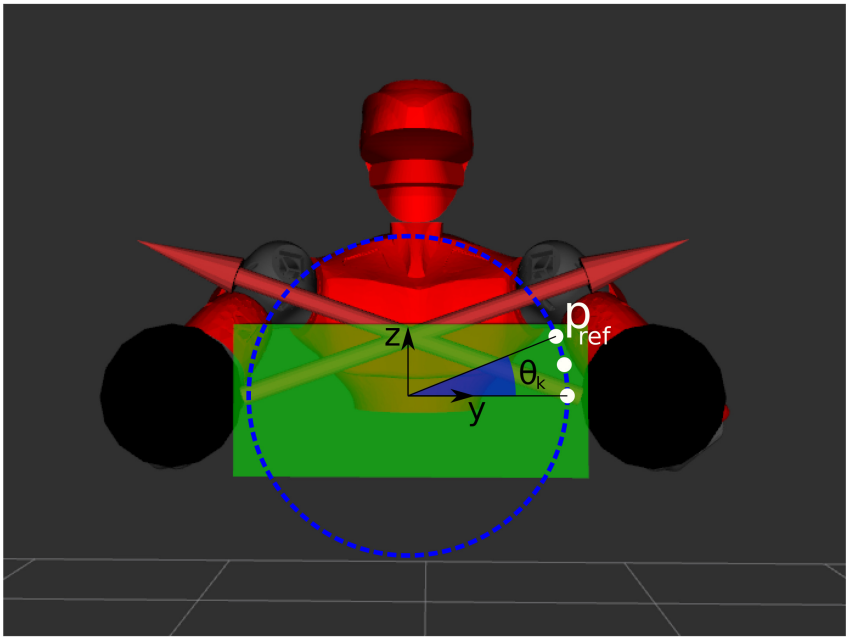


Figure 5.15: Circular trajectory to track

$$p_{ref} = \begin{bmatrix} x_{ini} \\ y_{ini} + R\cos(\frac{2\pi k}{N}) \\ z_{ini} + R\sin(\frac{2\pi k}{N}) \end{bmatrix} \tag{5.5}$$

Instead, $\mathbf{e}_{ok}$ is the box orientation error with respect to initial box orientation, express as RPY. Then, additional terms are regularization terms on the joint angles velocities and

global contact forces.

5.6.1 Result discussion

The NMPC ROS application starts with the initialization phase, which is the same as the payload holding problem; while the optimal control problem for the next prediction horizon is solved, the interpolator node feeds the robot to hold the box at its initial position. This task requires certain torque at the joint, Figure 5.18, which determines the motor heat up as shown in Figure 5.19.

Once the initialization phase is over, thanks to the recursive optimization problem solution the robot starts following the circular reference trajectory, as it might be seen in Figure 5.20. To execute the task the robot develops the torques in Figure 5.18 that contribute, together with the joint velocities in Figure 5.17, to heat up the left and right shoulder pitch. Figure 5.20, shows that both the temperature constrained and not temperature constrained optimal solution let the robot executing the required task until the 6^{th} NMPC iterations. According to Figure 5.19 the left and right would pass the bound after 100 seconds performing this task. The controller indeed intervenes modifying the robot configuration at run-time, moving the box closer to its pelvis and reducing the torque on the shoulder pitch and compensating with the two elbows, Figure 5.18. Figure 5.20 shows how the robot is able to continue performing the task finding a trade-off between thermal bound and task execution. Figures 5.23b, 5.23c depicts the difference between the two solutions, with the second one that continues performing the task with the box closer to the shoulders.

The same conclusion can be draw looking at Figure 5.16, which describes the joint angles evolution in both cases. As far as the end effector forces are concerned, Figure 5.21 shows the end effector local forces, which respect the friction cones and the payload constraints.

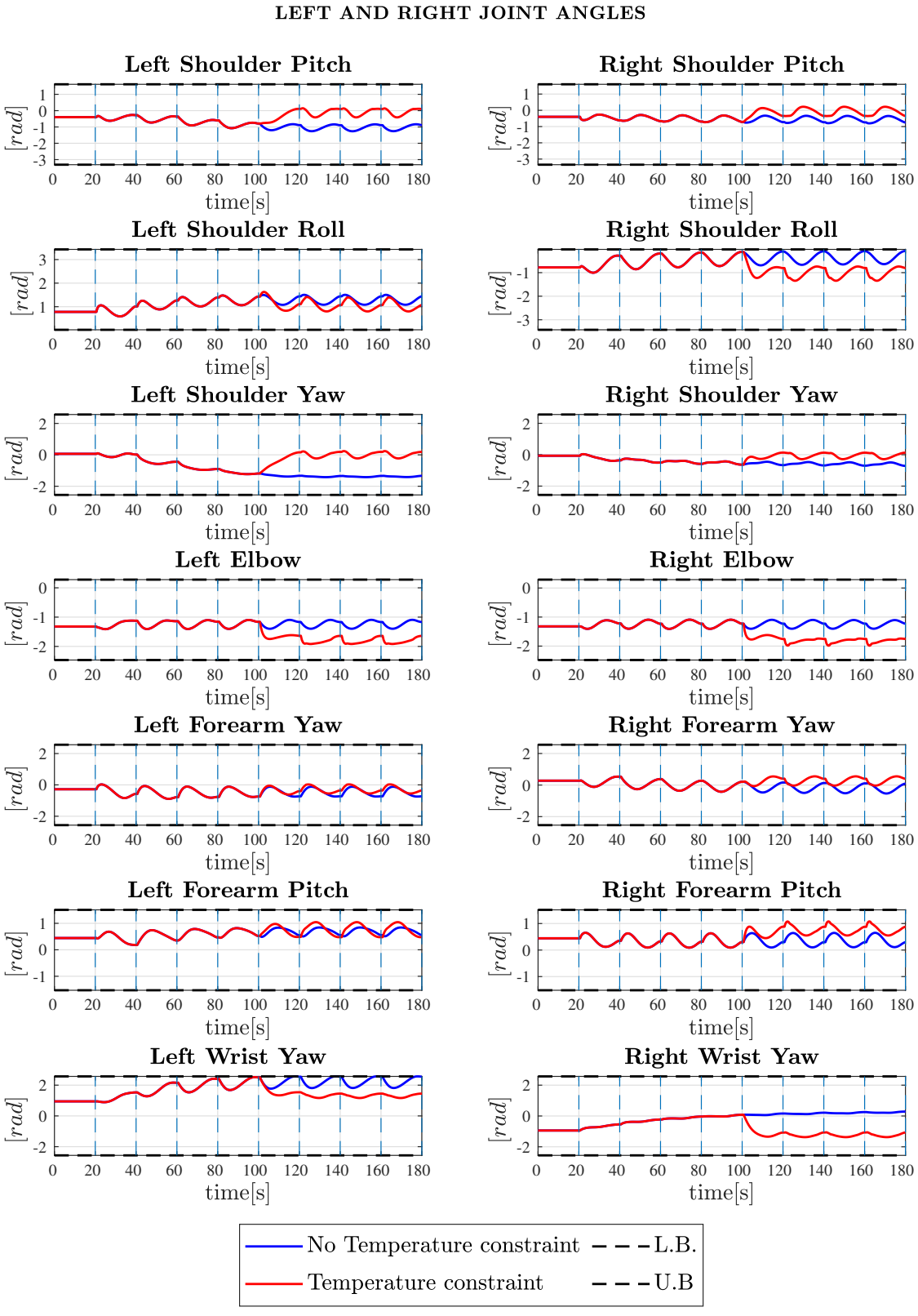


Figure 5.16: Circular trajectory tracking - Run-time joint angle comparison

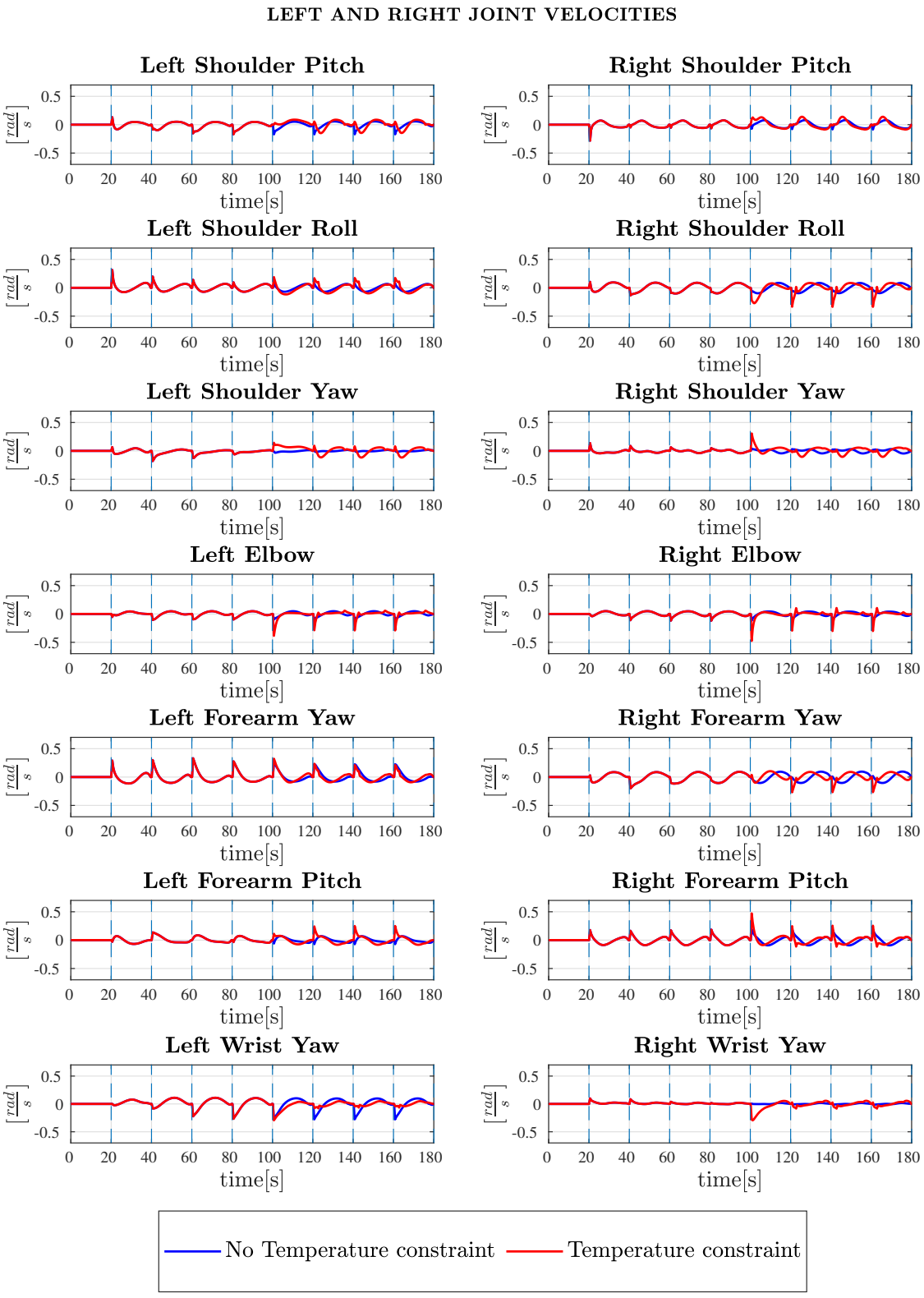


Figure 5.17: Circular trajectory tracking - Run-time joint velocities comparison

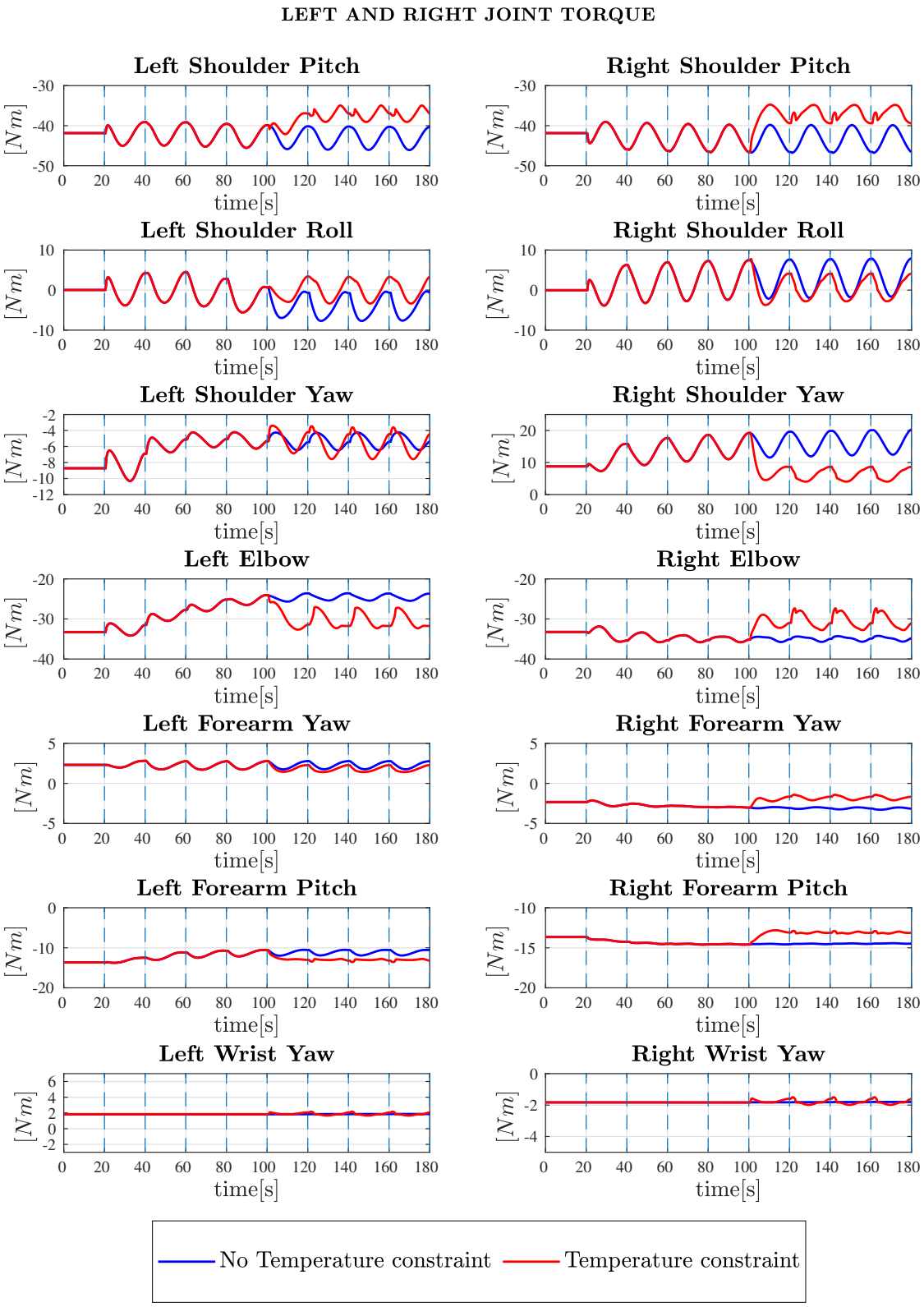


Figure 5.18: Circular trajectory tracking - Run-time joint torques comparison

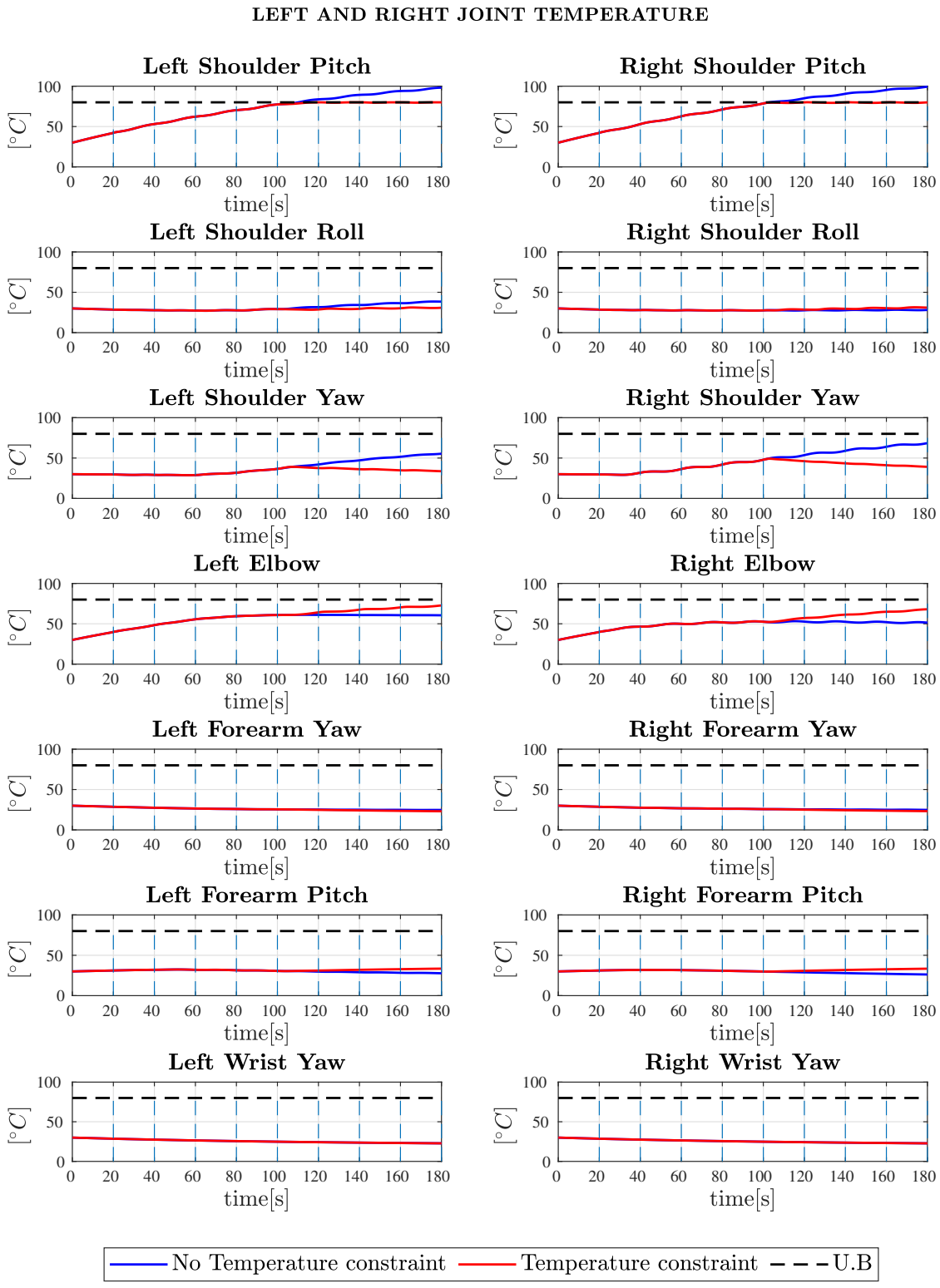


Figure 5.19: Circular trajectory tracking - Run-time joint temperatures comparison

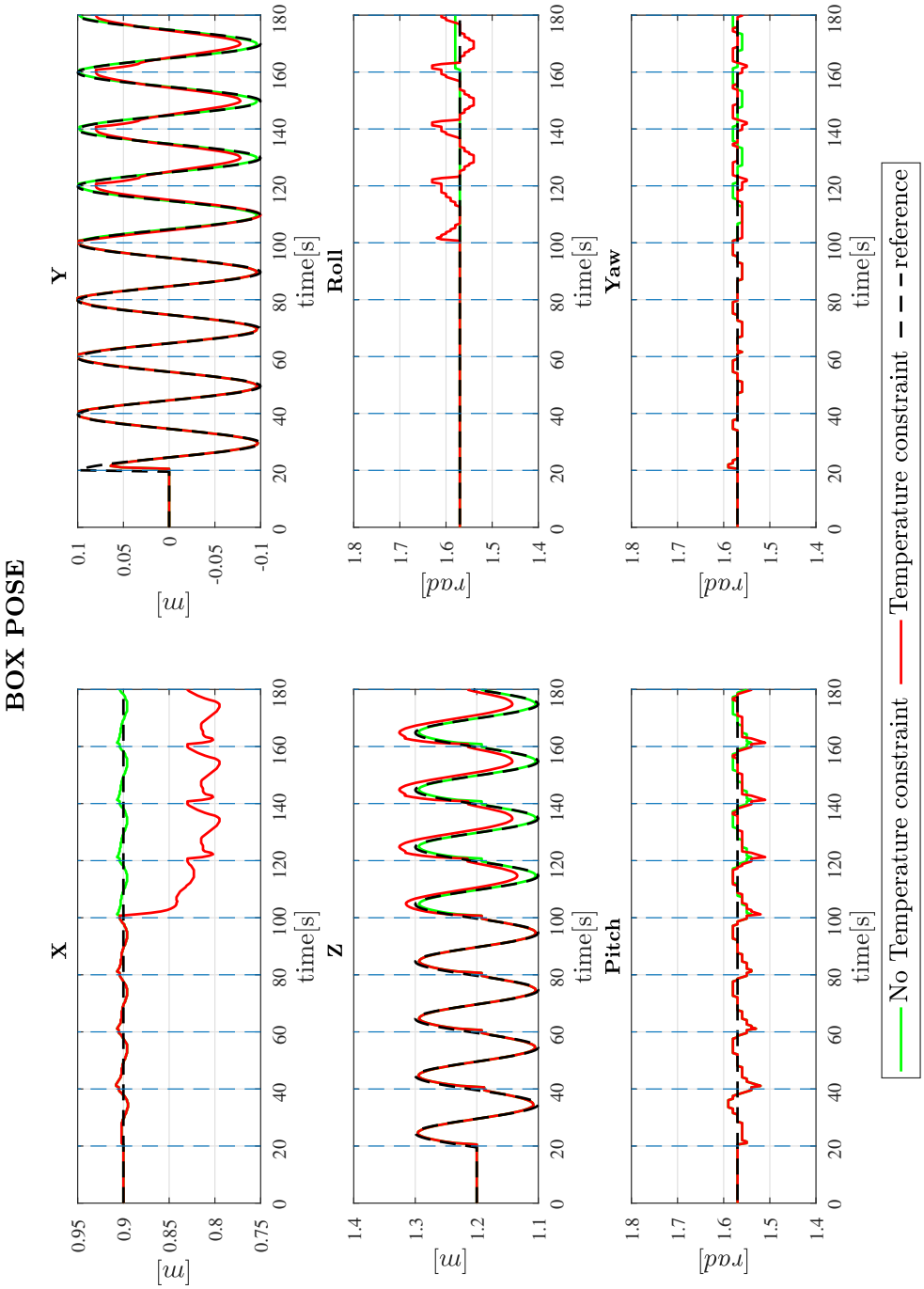


Figure 5.20: Circular trajectory tracking - Run-time box position and orientation comparison

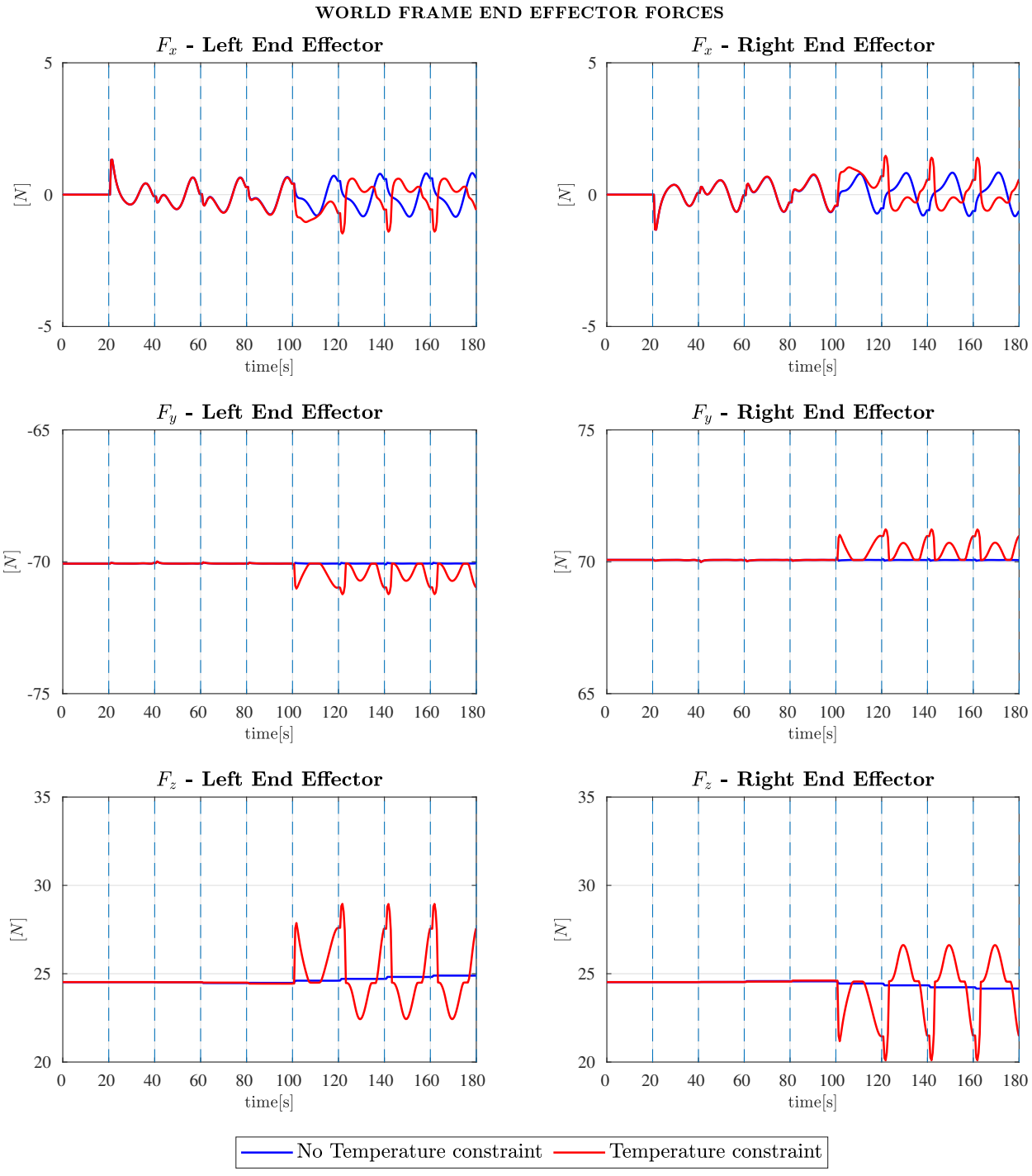


Figure 5.21: Circular trajectory tracking - Run-time global contact forces comparison

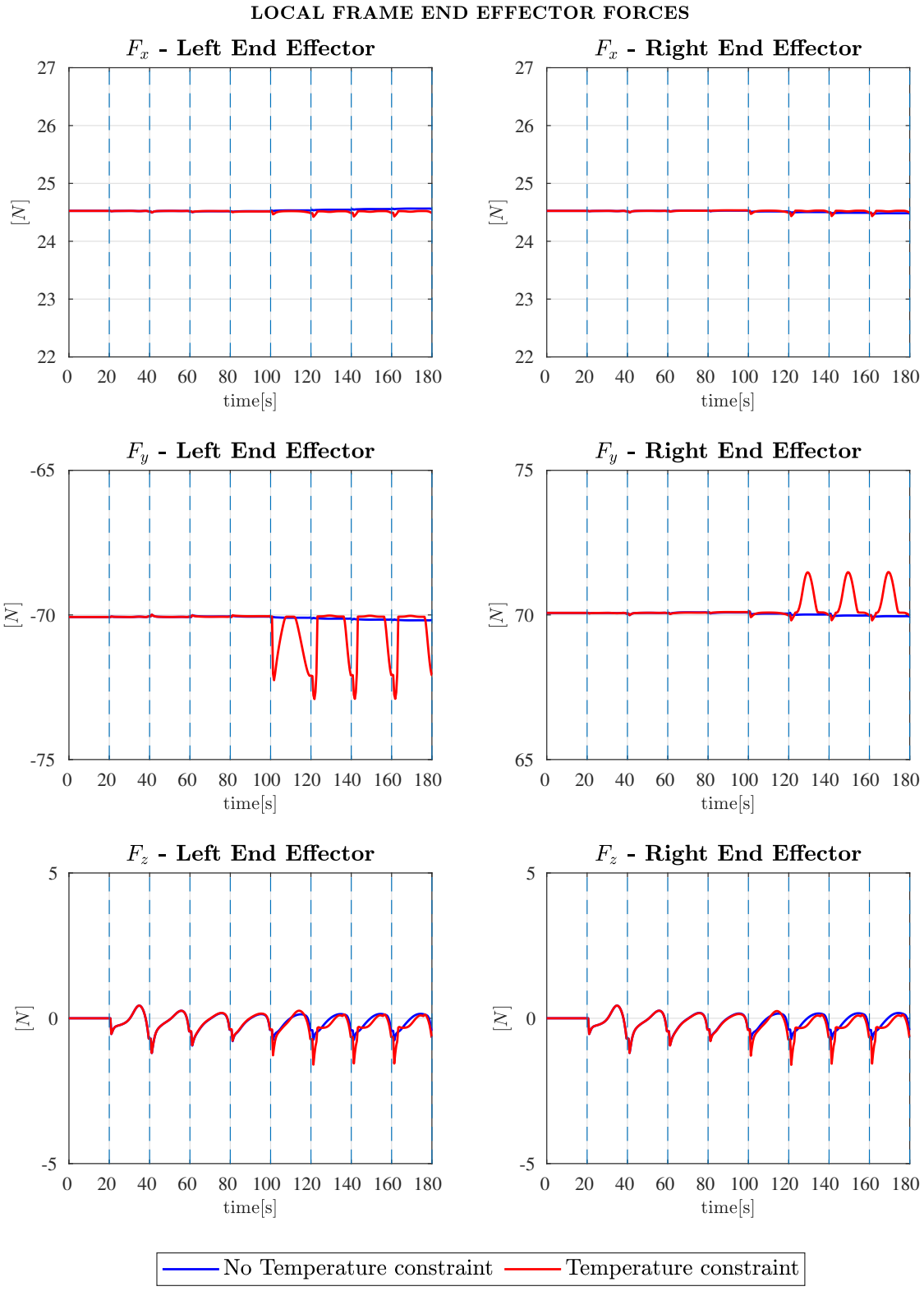
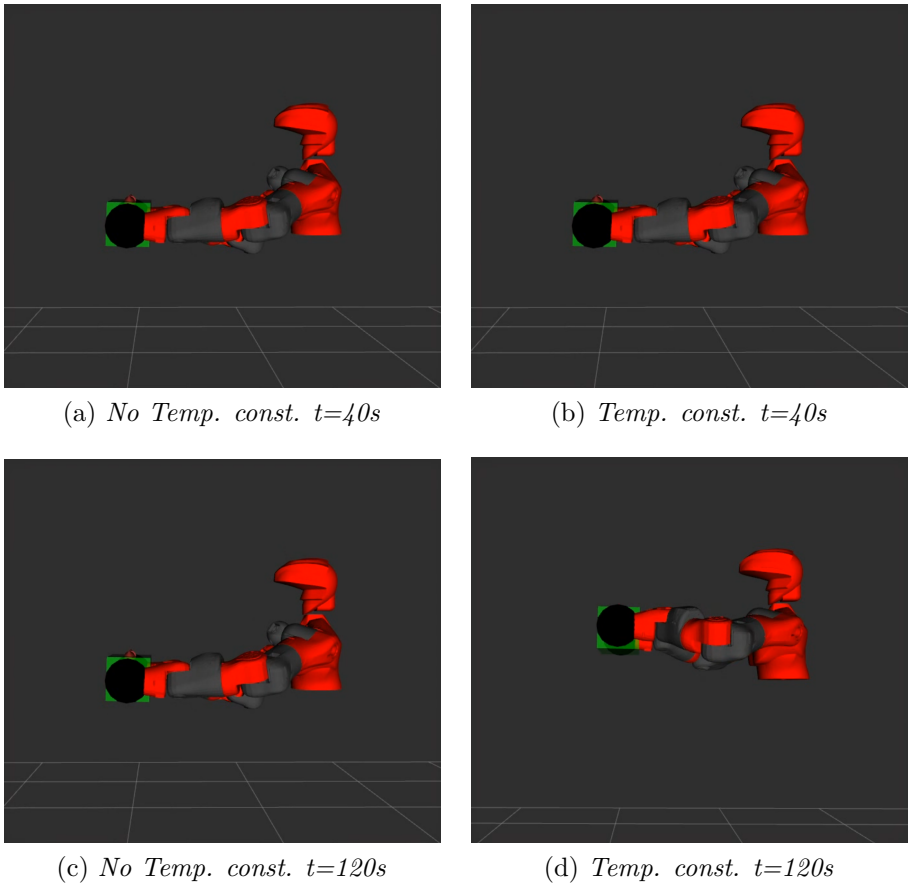


Figure 5.22: Circular trajectory tracking - Run-time local contact forces comparison



(a) *No Temp. const. t=40s* (b) *Temp. const. t=40s*

(c) *No Temp. const. t=120s* (d) *Temp. const. t=120s*

Figure 5.23: Circular trajectory tracking - Configuration evolution

Chapter 6

Conclusion

Over the past decade the possibility to employ robots in scenarios that are too dangerous for humans to enter, has significantly gained the interest of the robotics community. In particular, when such robots are used to perform heavy manipulation tasks they are likely to face thermal fatigue issues, which may eventually contribute to the robot damage. In this respect, this thesis has proposed a nonlinear model predictive (NMPC) approach to effectively prevent the motor burnout problem while executing a bimanual heavy manipulation task. To formulate the NMPC algorithm, the background on NMPC has been first provided, starting from the concepts of nonlinear programming and optimal control. Then, direct methods to deal with optimal control problems has been described both from a theoretical and practical standpoint. Subsequently, background knowledge on mathematical modelling of robotic manipulators has been provided, ranging from the definition of the end-effector pose to the analysis of the forward and inverse kinematic and dynamics of robots. During the study, it has been introduced the fixed-base bimanual manipulator platform over which the problem was formulated and then it has been formalized the optimal control problem describing a thermal bounded heavy payload manipulation in terms of optimization variables, cost function, and constraints. Finally, the complete optimal control problem has been used as starting point to define the model predictive controller. In the last Chapter, the description of the real robot platform, CENTAURO, and the software framework (CasADi, Pinocchio, ROS) exploited in this work have been introduced. Moreover, details of the algorithm implementation and experimental result summary have been provided. To conclude, the proposed nonlinear model predictive controller has been tested in two heavy payload manipulation tasks. The first test required the robot to hold a box in a fixed position in Cartesian space. The result shows that, while executing the assigned task, the robot configuration was autonomously changed at run-time in order not to violate the thermal bound imposed to protect motors from burnout. The first test solution was compared to a minimum effort problem solution, showing that thanks to the NMPC algorithm, the torque reduction was temperature dependent and considers thermal fatigue of specific joints, whereas a minimum effort problem minimized the torques independently of the temperature of the motors. A similar result was shown in the second experiment, which required the

robot to move the box center of mass over a circular trajectory at a fixed distance from its pelvis. In that case, the robot executes the task until continuing the task execution would require the temperature constraint violation. In the second test, the robot autonomously moves the box closer to its pelvis finding a new equilibrium position that protects the motors from burnout. Our results show that a robot can perform high-load tasks while preventing motors from overheating during operation through joint reconfiguration; naturally, other joints must carry more load to produce the desired task in the Cartesian space. As a result, it is possible to extend the working time of the robot during tasks and avoid any potential damage due to the overheating of the actuators.

During the NMPC algorithm tests, some problems were pointed out. We recall that at each time step the NMPC solves an optimal control problem that predicts the system evolution over the future user-defined time horizon t_h. The NMPC method makes sense if any optimal control problem requires a computation time t_c shorter than the prediction horizon. Moreover, the solution should be feasible and its feasibility is strictly related to, the prediction horizon and the number of multiple shooting nodes. Increasing the number of multiple shooting nodes would increase the computation time needed to solve the optimal control problem as well, whereas reducing them too much would decrease the solver degrees of freedom resulting in the problem infeasibility, unless the prediction horizon is reduced as well. Having a short time horizon, though, makes it difficult to satisfy the time constraint $t_c \leq t_h$. This works used a large time horizon taking advantage of the slow motor thermal dynamics and a limited number of multiple shooting nodes in order to respect the time constraint. The aforementioned considerations show one drawback of this approach, which requires finding a proper time horizon, number of nodes and cost function weight empirically for each different task to perform.

6.1 Future research directions

The natural evolution of this thesis would move in the direction of parameters identification for the employed thermal model. In fact, as mentioned in Chapter 5, the thermal model, employed to predict the motor temperature evolution, does not represent the real motor thermal model because of the unknown parameters. The experiments on the real CENTAURO platform performed in Chapter 5 allowed the collection of real sampled data that will be used in the future for the identification procedure. To this end, due to the linearity of the thermal model with respect to the unknown parameters, the least-squares method will be likely employed. The objective of the method is to estimate the parameters of the model, based on the observed pairs of values. The best fit in the least-squares sense minimizes the sum of squared residuals, which is defined as the difference between an observed value, and the fitted value provided by a model.

From a long-term perspective, a second future work will consist in moving towards the

whole body optimization by considering also the lower body of CENTAURO. In this thesis, the optimization process was designed taking into account only two arms of the CENTAURO platform. To fully exploit the fatigue management system, the method may be applied including also the CENTAURO legs articulations, wheels and torso joint motions.

Appendix A

Centauro platform limits

A.1 Joint limits

Joint name	**Angle L.B.[rad]**	**Angle U.B.[rad]**	**Velocity**[$\frac{\mathbf{rad}}{\mathbf{s}}$]	**Torque**[**Nm**]
Left Shoulder pitch	−3.312	1.615	3.86	147.0
Left Shoulder Roll	0.020	3.431	3.86	147.0
Left Shoulder yaw	−2.552	2.566	6.06	147.0
Left Elbow	−2.465	0.280	6.06	147.0
Left Forearm Yaw	−2.569	2.562	11.72	55.0
Left Forearm Pitch	−1.529	1.509	11.72	55.0
Left Wrist Yaw	−2.565	2.569	20.35	28.32
Right Shoulder pitch	−3.3458	1.6012	3.86	147.0
Right shoulder Roll	−3.4258	−0.0138	3.86	147.0
Right shoulder yaw	−2.5614	2.5606	6.06	147.0
Right Elbow	−2.4794	0.2886	6.06	147.0
Right Forearm Yaw	−2.5394	2.5546	11.72	55.0
Right Forearm Pitch	−1.5154	1.5156	11.72	55.0
Right Wrist Yaw	−2.5554	2.5686	20.35	28.32

Appendix B

Test parameters

B.1 Payload holding experiment

Parameter	**Symbol**	**Value**	**Units**
Box holding position	p_{box}	$[0.8, 0, 1.0]$	m
Prediction Horizon	t_h	20	s
Multiple shooting nodes	N	40	#
Number of iterations	j	5	#
Position error weight	w_p	1000.0	#
Contact force weight	w_f	0.01	#
Joint velocity weight	w_q	10.0	#
Torque weight	w_τ	1.0	#
Payload length	L	0.35	m
Payload mass	m	10	kg
Friction coefficient	μ	0.35	#
Temperature bound	T_{bound}	80	$^\circ C$
Room temperature	T_a	20	$^\circ C$

B.2 Circular trajectory tracking experiment

Parameter	**Symbol**	**Value**	**Units**
Box initial position	p_{ini}	$[0.9, 0, 1.2]$	m
Prediction Horizon	t_h	20	s
Multiple shooting nodes	N	40	#
Number of iterations	j	5	#
Position error weight	w_p	1000.0	#
Orientation error weight	w_o	100.0	#
Contact force weight	w_f	0.01	#
Joint velocity weight	w_q	10.0	#
Payload length	L	0.35	m
Payload mass	m	5	kg
Trajectory radius	R	0.1	m
Friction coefficient	μ	0.35	#
Temperature bound	T_{bound}	80	$^{\circ}C$
Room temperature	T_a	20	$^{\circ}C$

Bibliography

[1] Dana Kulic and Elizabeth Croft. "Pre-collision safety strategies for human-robot interaction". In: *Auton. Robots* 22 (Jan. 2007), pp. 149–164. DOI: 10.1007/s10514-006-9009-4.

[2] A. D. Luca et al. "Collision Detection and Safe Reaction with the DLR-III Lightweight Manipulator Arm". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 1623–1630. DOI: 10.1109/IROS.2006.282053.

[3] Sami Haddadin et al. "On making robots understand safety: Embedding injury knowledge into control". In: *The International Journal of Robotics Research* 31.13 (2012), pp. 1578–1602. DOI: 10.1177/0278364912462256.

[4] S. Trujillo and M. Cutkosky. "Thermally constrained motor operation for a climbing robot". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 2362–2367. DOI: 10.1109/ROBOT.2009.5152870.

[5] L. Peternel, N. Tsagarakis, and A. Ajoudani. "A Method for Robot Motor Fatigue Management in Physical Interaction and Human-Robot Collaboration Tasks". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 2850–2856. DOI: 10.1109/IROS.2018.8594196.

[6] Matthieu Guilbert, Pierre-Brice Wieber, and Luc Joly. "Optimization of Complex Robot Applications under Real Physical Limitations". In: *The International Journal of Robotics Research* 27 (May 2008). DOI: 10.1177/0278364908090465.

[7] I. Kumagai et al. "Whole body joint load reduction control for high-load tasks of humanoid robot through adapting joint torque limitation based on online joint temperature estimation". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 463–468. DOI: 10.1109/HUMANOIDS.2014.7041402.

[8] S. Noda et al. "Online maintaining behavior of high-load and unstable postures based on whole-body load balancing strategy with thermal prediction". In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. 2014, pp. 1166–1171. DOI: 10.1109/CoASE.2014.6899474.

[9] J. Urata et al. "Thermal control of electrical motors for high-power humanoid robots". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 2047–2052. DOI: 10.1109/IROS.2008.4651110.

[10] M. Guilbert, P. Wieber, and L. Joly. "Optimal Trajectory Generation for Manipulator Robots under Thermal Constraints". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 742–747. DOI: 10.1109/IROS.2006.282623.

[11] Wei Tan et al. "Trajectory Optimization for High-Power Robots with Motor Temperature Constraints: 19th Annual Conference, TAROS 2018, Bristol, UK July 25-27, 2018, Proceedings". In: (July 2018), pp. 3–14. DOI: 10.1007/978-3-319-96728-8_1.

[12] J. Lu and A. Murray. "DSP-based thermal protection for brushless servo motor". In: *IEE Colloquium on Variable Speed Drives and Motion Control*. 1992, pp. 7/1–7/4.

[13] O. Craiu et al. "3D Finite Element thermal analysis of a small power PM DC motor". In: *2010 12th International Conference on Optimization of Electrical and Electronic Equipment*. 2010, pp. 389–394. DOI: 10.1109/OPTIM.2010.5510335.

[14] E. Chauveau et al. "A statistical approach of temperature calculation in electrical machines". In: *IEEE Transactions on Magnetics* 36.4 (2000), pp. 1826–1829. ISSN: 1941-0069. DOI: 10.1109/20.877800.

[15] R. Yabiku et al. "Use of Thermal Network on Determining the Temperature Distribution Inside Electric Motors in Steady-State and Dynamic Conditions". In: *IEEE Transactions on Industry Applications* 46.5 (2010), pp. 1787–1795. ISSN: 1939-9367. DOI: 10.1109/TIA.2010.2057398.

[16] "1. Introduction to Nonlinear Programming". In: *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, pp. 1–49.

[17] Lorenz Biegler. "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming* 106 (Jan. 2004).

[18] Paul Boggs and Jon Tolle. "Sequential Quadratic Programming". In: *Acta Numerica* 4 (Jan. 1995), pp. 1–51. DOI: 10.1017/S0962492900002518.

[19] Moritz Diehl and Sébastien Gros. *Numerical Optimization of Dynamic Systems*. 2016.

[20] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846286417.

[21] Rodrigo S. Jamisola and Rodney G. Roberts. "A more compact expression of relative Jacobian based on individual manipulator Jacobians". In: *Robotics and Autonomous Systems* 63 (2015), pp. 158 –164. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2014.08.011. URL: http://www.sciencedirect.com/science/article/pii/S0921889014001699.

[22] Luca Muratore et al. "XBotCore: A Real-Time Cross-Robot Software Platform". In: Apr. 2017. DOI: 10.1109/IRC.2017.45.

[23] A. Laurenzi et al. "CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 591–596. DOI: 10.1109/ICRA.2019.8794464.

[24] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[25] Justin Carpentier et al. "The Pinocchio C++ library — A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: Jan. 2019. DOI: 10.1109/SII.2019.8700380.