Universidade de Brasília
Faculdade de Tecnologia

Kinematic Predictive Control for Trajectory of a Robotic Manipulator

Lucas de Moura Quadros

TRABALHO DE GRADUAÇÃO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasilía

2022

Universidade de Brasília
Faculdade de Tecnologia

Kinematic Predictive Control for Trajectory of a Robotic Manipulator

Lucas de Moura Quadros

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Geovany Araújo Borges

Brasilía
2022

Moura Quadros, Lucas de.
M929k Kinematic Predictive Controlfor Trajectory of aRobotic Manipulator / Lucas de Moura Quadros; orientador Geovany Araújo Borges. -- Brasilía, 2022.
74 p.

Trabalho de Graduação em Engenharia de Controle e Automação -- Universidade de Brasília, 2022.

1. kinematics. 2. predictive. 3. control. 4. robot. I. Araújo Borges, Geovany, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Kinematic Predictive Control for Trajectory of a Robotic Manipulator

Lucas de Moura Quadros

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Trabalho aprovado. Brasilía, 11 de Maio de 2022:

Prof. Dr. Geovany Araújo Borges,
UnB/FT/ENE
Orientador

Guilherme Caribe de Carvalho,
UnB/FT/ENM
Examinador interno

José Maurício Santos Torres da Motta,
UnB/FT/ENM
Examinador interno

Brasilía
2022

Dedico este trabalho à minha mãe Valéria Maria
e ao meu pai, Luiz Alberto.

Acknowledgements

A Universidade de Brasília me proporcionou muitos aprendizados, no quesito técnico acadêmico mas também em relação a questões interpessoais, sociais, políticas e até filosóficas. Penso que esse é o intuito de se haver a Universidade, inclusiva, que valoriza a ciência e a formação de seus alunos. Foi possível conhecer várias pessoas diferentes, em termos de personalidade, estilos de vida e professores com diferentes metodologias, abordagens.

Ao Professor Geovany, sou grato por vários ensinamentos sobre robótica inclusive antes de ter me orientado, da sua forma serena e inteligente de lecionar. À professora Mariana, agradeço bastante por ter se disposto de tão boa vontade de ter aceitado me orientar em meio à pandemia de Covid-19, por ter sido sempre tão cordial, atenciosa e assertiva nos seus ensinamentos e proposições. Mesmo no ensino à distância, pude ficar bem confortável para sanar diferentes dúvidas e explicar origens de certos desafios encontrados.

Ao colega de área Marcos Pereira, que me orientou em vários aspectos sobre como trabalhar com diferentes ferramentas necessárias para o trabalho, pela sua simpatia, inteligência e postura eloquente em ajudar a resolver problemas.

Aos meus colegas e amigos com quem pude ter a oportunidade de conviver ao longo da graduação, também agradeço bastante pela companhia e pelos aprendizados de pessoas astutas e inteligentes que puderam me ensinar diferentes modos de pensar e agir.

Abstract

Robotics is a very wide field that involves diverse engineering types, computer science, mathematics and so on. There are multiple tools that enable to work with robotics, even in simulation environments. One important aspect of a robotic system is the fulfillment of a geometrical task, in a specific period of time. Even if the assignment is not feasible within the requested time, it is important to maintain the satisfaction of the geometrical task.

In this context, this work consists in exert the kinematic control of a redundant robot manipulator, using dual quaternion algebra. Two kinds of controllers were developed: one is known as the Local method, which only considers information related to the current state of the robot, while the other takes into account the future evolution of the task in order to compute the control law.

A scaling factor was also considered, which enables to make the task more flexible in terms of the total time requested to satisfy it. Both the kinds of controllers applied the scaling factor, in which for one of the two trajectories tested, the Model predictive control (MPC) produced a higher scaling factor, while for the other one, the Local method provided a lower scaling factor. Different physical quantities were also taken into account and compared. It was possible to analyze the advantages of the MPC controller towards the Local method.

Without considering a scaling factor, a simpler MPC controller was developed, in order to compare its performance with a proportional controller based in a different error metric.

Keywords: kinematic, control, robot, manipulator, predictive

List of Figures

List of Tables

Contents

1 Introduction

The impetus of working with robots began to gain strength in the beginning of the last century (DE ALMEIDA, n.d.), motivated by the urge to increase productivity and upgrade product quality. Hence, right in that epoch, the primary applications for the industrial robots were originated. George Devol(1912-2011) was the one who founded the industrial robotics, as e invented the first industrial robot called Unimate [1], which was installed at the Ford Motor Company plant in Trenton (New Jersey) in 1961 [2]. Thanks to countless capabilities that embedded and micro-controlled systems provide, the field of robotics is, nowadays, going through a period of continuous growth that will enable, in a near future, the implementation of robots with capacity to predict situations and act properly.

Robotics consists in a technological and educational discipline that manages systems composed of mechanical parts usually combined together with integrated circuits, making motorized mechanical systems controlled by electronic components and computational systems. Robotics is the object of study in several areas: computing, aerospace, mechanics, automation, electrical, control theory, etc.

The everyday use of robotics is growing, such as robot vacuum, and robots for medical surgery (SANTOS; LEME; STEVAN, 2018). This applied science, nowadays used by innumerable factories and segments of industries, has generally been efficient in handling issues such as in reducing cost, improving productivity and alleviating labor problems (CRAIG, 2005). However, even considering these advantages, the employment of robotics causes many problems in the social context of society, such as structural unemployment (REIS, n.d.). Nevertheless, robotic manipulators are made to make life easier and can be well used to replace repetitive jobs, thus directing people to jobs that demand more human qualities.

1.1 Contextualization

A robot manipulator is a set of bodies connected by joints, forming kinematic chains that define a mechanical structure. The manipulator includes the actuators, which act on the mechanical structure, modifying its configuration, and the transmission, which connects the actuators to the structure. The terms manipulator and robot are often used for the same purpose, although formally this is not precise. Nevertheless, in the context of this project, the robot is a manipulator.

There are manifold robot manipulator applications nowadays that, in a near future,

[1] https://www.history101.com/unimate-first-industrial-robot/

[2] https://www.thehenryford.org/collections-and-research/digital-collections/artifact/373806/#slide=gs-253931

may depend upon precise tracking of a pre-specified continuous path. Typical cases of applications in such sense consists in seam tracking, arc welding, cutting (laser and water jet), spray painting, contours inspection, coordinated parts transfer and manufacturing operations. The geometrical tasks are normally defined in relation to the end effector of the robotic manipulator and may establish trajectories as a function of time (which is the usual), but also as a function of position. There are some trouble of fulfilling this request of temporal path tracking:

- significant nonlinearities are present in the dynamics and geometry;
- unidentified parameters, modeling and measurement errors;
- unplanned changes in operating conditions;
- other disturbances may affect the robot;

These issues may be a great challenge in tuning an accurate control for the robot involved.

To fulfill this task of accurate path following, an efficient control strategy is needed, which implies the following aspects:

- accurate tracking the specified trajectory of the end effector, usually as a function of time;
- minimal complexity as possible, to enable fast computation and a high sampling rate for the control system;
- reliability, specially in regard of robustness of the control methodology.

It is important to satisfy the geometrical task that is required to the robot, even if it is not possible to fulfill it in the specified time. To produce a large discrepancy in relation to the original task might cause harm to the manipulator itself or be dangerous for people around it.

1.2 Problem definition

For a given task assigned to a robotic manipulator, a temporal function of a pose is defined, in the set of dual quaternions, that the end effector must fulfill. However, depending on the limitations of the actuators, the assignment of the trajectory in the specified time may not be feasible. Physical restrictions can be the range of joint angles, joint speed or maximum torque that the actuator can exert.

1.3 Project Goal

The objective of this work is to control the trajectory of the end effector, in order to preserve the desired geometric path, even if the trajectory is not feasible in the pre-established time. To satisfy this task, two different kinematic controllers were tested, taking into account the robot capabilities to determine the control law.

One of the methods used was named Local method, since it only considers the current configuration of the robot to compute the control law. The alternate method is known as Model Predictive Control (MPC), since it considers a kinematic model to make predictions of the manipulator's state to determine the control signal.

The robot involved is presented in a simulation environment called CoppeliaSim and the control methods were developed in Matlab, wherein these programs exchange data through a tool called ROS. The objective is to test the controllers in a simulation before, in future works, implement it in the real robot Meka A2, that is located at LARA in the University of Brasilia.

1.4 Results

For the two trajectories worked in this project, different comparisons are made, in regard to fulfillment of the task in the specified time, the level of deformation of the path, the torques applied, the computed control actions, scaling factor and even the joint angular positions measured along the trajectory. Many graphs were constructed to enable a qualitative and quantitative analysis.

The results have shown the effectiveness of the MPC controller in relation to the other purely local method, specially for the one MPC with more prediction instants. It was not easy, however, to explore the nuances that permitted to achieve plausible results with the MPC. A comparison was made between the MPC controller and a proportional method based on the Moore-Penrose pseudo-inverse matrix of the robot pose Jacobian.

1.5 Text Organization

The progress of this text consists in:

1. presentation of theoretical concepts necessary for the understanding of the work, such as the algebra of dual quaternions, the paradigm of differential kinematic controlling and model predictive control;

2. description of the three frameworks used in this project and how they are used: the robot simulator named CoppeliaSim (ROHMER; SINGH, S. P. N.; FREESE, 2013),

the program for mathematical computations known as Matlab (MATLAB, 2010) and the software needed to communicate the nodes, which is called ROS (STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL., 2018) (Robot Operational System);

3. exhibition and discussion of the results obtained through different controllers, in the two tested trajectories.

4. conclusions and suggestions of future works.

2 Mathematical Foundations

In this chapter, it is presented some mathematical foundations that enable to understand how to use the dual quaternions to represent rigid motions in a three-dimensional space. This set of numbers was proved to be very convenient for the matters of this project. It is also explained some fundamentals in regard to kinematic controlling.

2.1 Quaternions

In 1843, the mathematician William Rowan Hamilton created the quaternions numbers, which have three imaginary units $\hat{\imath},\hat{\jmath},\hat{k}$, wherein the quaternionic unit $\hat{\imath}$ is equivalent to the one defined for complex numbers (ADORNO, 2017). As a matter of fact, the quaternions may be considered as an extension of complex numbers, wherein the imaginary units $\hat{\jmath},\hat{k}$ are analogous to $\hat{\imath}$, presenting the properties (HAMILTON, 1844):

$$\hat{\imath}^2 = \hat{\jmath}^2 = \hat{k}^2 = \hat{\imath}\hat{\jmath}\hat{k} = -1 \tag{2.1}$$

The real and imaginary parts of a quaternion are orthogonal to each other, wherein $\hat{\imath},\hat{\jmath},\hat{k}$ can be regarded as unit versions of a right handed coordinate frame, in which can be applied the same convention as for the cross product. The formal definition of the set $\mathbb{H}$ of quaternions is given by (ADORNO, 2017):

$$\mathbb{H} \triangleq \{q_1 + q_2\hat{\imath} + q_3\hat{\jmath} + q_4\hat{k} \,:\, q_1,q_2,q_3,q_4 \in \mathbb{R}\} \tag{2.2}$$

The scalar q_1, named as Re ($\boldsymbol{q}$), is the real part of a quaternion $\boldsymbol{q}$. The part that contains the imaginary elements is given by Im($\boldsymbol{q}$) = $q_2\hat{\imath} + q_3\hat{\jmath} + q_4\hat{k}$. Therefore, the quaternion can be expressed as q = Re (q) + Im (q), whereas its conjugate is defined by q^* = Re (q) - Im (q) (ADORNO, 2017).

The usual operations of sum and subtraction are simply defined as, respectively, a sum/subtraction of each term in each coordinate of the quaternion numbers involved, as exhibited in (ADORNO, 2017). As for the quaternion multiplication, it can be applied the distributive property, wherein the actual operation definition is also shown in (ADORNO, 2017).

It is pertinent to define a quaternion norm, which is given by $||q|| \triangleq \sqrt{qq^*}$. The norm is useful to obtain a quaternion inverse and also there are cases in which quaternions with unitary norm have special properties. A quaternion inverse is determined as (ADORNO, 2017):

$$q^{-1} = \frac{q^*}{||q||^2} \tag{2.3}$$

which, of course, is only valid if the norm is not zero.

An operation that can be important is the one known as vec_4, which extracts the quaternion coordinates, transforming it into a vector; for a quaternion $q = q_1 + q_2\hat{i} + q_3\hat{j} + q_4\hat{k}$ (ADORNO, 2017):

$$vec_4(q) = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{2.4}$$

It is also relevant to highlight the existence of the Hamiltonian operators $\overset{+}{H}_4(\cdot)$ and $\overset{-}{H}_4(\cdot)$, which are defined as (ADORNO, 2017):

$$\overset{+}{H}_4(q) = \begin{bmatrix} q_1 & -q_2 & -q_3 & -q_4 \\ q_2 & q_1 & -q_4 & q_3 \\ q_3 & q_4 & q_1 & -q_2 \\ q_4 & -q_3 & q_2 & q_1 \end{bmatrix}, \qquad \overset{-}{H}_4(q') = \begin{bmatrix} q'_1 & -q'_2 & -q'_3 & -q'_4 \\ q'_2 & q'_1 & q'_4 & -q'_3 \\ q'_3 & -q'_4 & q'_1 & q'_2 \\ q'_4 & q'_3 & -q'_2 & q'_1 \end{bmatrix} \tag{2.5}$$

wherein these represented matrices have the following property:

$$\begin{aligned} vec_4(qq') &= \overset{+}{H}_4(q)vec_4(q') \\ &\equiv \overset{-}{H}_4(q')vec_4(q) \end{aligned} \tag{2.6}$$

It can be useful to work with these defined operators in equations 2.4 and 2.5 to do some procedures, some calculations needed in a control system with representations in quaternions.

2.1.1 Translation

A specific type, called pure quaternions (which have null real part), is associated to the representation of translation movements in a three dimensional space (ADORNO, 2017). In figure 1, it is illustrated a pure quaternion $\boldsymbol{p}$, wherein each coordinate is shown as a projection in the respective axis.

It will be shown later, in the section dedicated to dual quaternions, how exactly pure quaternions are used to compute a translation in the space.

2.1.2 Rotation

It is known that complex numbers enable to perform rotations of a two-dimensional vector in a plane. Similarly, the vectors in three dimensions can be submitted to an analogous operation, through the use of unit quaternions, that form a group of rotations $SO(3)$ inside a sphere (FIGUEREDO, 2016).

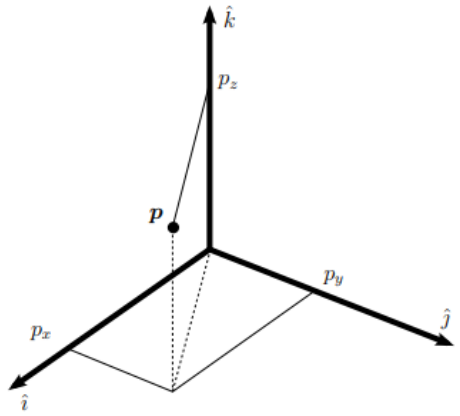

Figure 1 – Imaginary quaternion $p = p_x\hat{\imath} + p_y\hat{\jmath} + p_z\hat{k}$ (ADORNO, 2017)

Similarly to the fact that complex numbers enable to perform rotations in a plane, unit quaternions form the group of three-dimensional rotations $SO(3)$ in the sphere (FIGUEREDO, 2016). The unit quaternion $r = \cos(\theta/2) + n\sin(\theta/2)$ represents a rotation of an angle θ around the axis $n = n_x\hat{\imath} + n_y\hat{\jmath} + n_z\hat{k}$ (KUIPERS, 1999), which is illustrated in figure 2. It is possible to show that the imaginary quaternion n is also unitary:

$$\begin{aligned} ||r||^2 = 1 = \cos^2(\theta/2) + \sin^2(\theta/2)\,(n_x^2 + n_y^2 + n_z^2) \\ \Rightarrow 1 = 1 - \sin^2(\theta/2) + \sin^2(\theta/2)\,(n_x^2 + n_y^2 + n_z^2) \\ \Rightarrow \sin^2(\theta/2)(n_x^2 + n_y^2 + n_z^2 - 1) = 0 \\ \therefore n_x^2 + n_y^2 + n_z^2 = ||n||^2 = 1 \end{aligned} \tag{2.7}$$

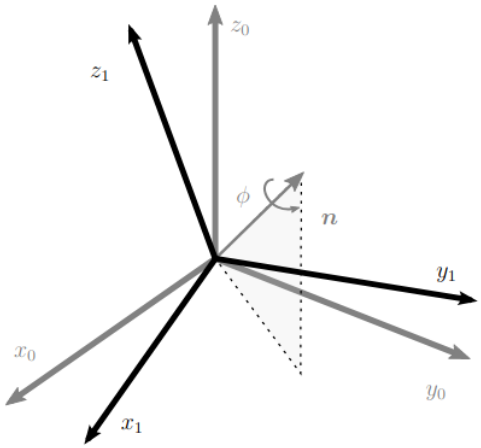

Figure 2 – Rotation angle θ around the unit rotation axis $\boldsymbol{n}$ (ADORNO, 2017)

As regard of unit quaternions, the conjugate $r^* = \cos(\theta/2) - \sin(\theta/2)n$ performs the inverse operation: rotation of an angle $-\theta$ around the axis n. As a matter of fact, for any unit quaternion q, it is valid $qq^* = 1$, which makes, by definition, that $q^* = q^{-1}$.

A point or a vector can be represented through distinct frames. Considering a point p, the representations in quaternions in relation to the coordinate frames $\mathcal{F}^0$ and $\mathcal{F}^1$ are

given, respectively, by the pure quaternions p^0 and p^1. Considering, for example, that the frame $\mathcal{F}^1$ is originated by a a rotation of $\mathcal{F}^0$, through a quaternion r_1^0, the point p can then be expressed, in relation to $\mathcal{F}^1$, as (KUIPERS, 1999):

$$p^1 = r_0^1 p^0 r_0^{1^*} \tag{2.8}$$

The conversion in equation 2.8 is known as frame rotation (FIGUEREDO, 2016), illustrated in figure 3.

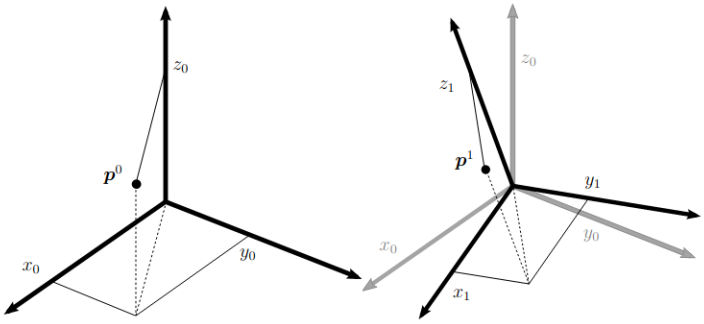

Figure 3 – Illustration of frame rotation (ADORNO, 2017)

In regard of the same transformation, but with another point of view, it is considered p_0^0 to be a vector expressed in relation to a coordinate frame $\mathcal{F}_0$. A rotation of p_0^0 can then be represented with respect to the same frame (KUIPERS, 1999):

$$p_1^0 = r_1^0 p_0^0 r_1^{0^*} \tag{2.9}$$

The transformations of frame and vector rotation differ solely in a matter of geometric interpretation, as the calculations are equal. For the frame rotation, the point remains immobile while the frame moves. Alternatively, for the rotation of a point, the coordinate frame remains static while the vector rotates (FIGUEREDO, 2016).

The difficulties that emanate from Euler angle/axis representations can be overcome with the usage of the unit quaternions. A rotation of an angle $-\theta$ around an axis $-n$ yields to the same quaternion corresponding to a rotation θ around the axis n. This property is a solution to the non-uniqueness problem of the angle-axis representation (B. SICILIANO L. SCIAVICCO; ORIOLO, 2009). However, the unit quaternion representation is subject to the unwinding phenomenon wherein the attitude of the rigid body may diverge to the antipodal representation, generating needless rotation. This problem presents topological issues (SILVA PEREIRA, 2016).

Rotation quaternions are, hence, a four-parameter representation with unit norm, which restrains it to three degrees of freedom actually. This set presents a particular algebra and permits to compute rotations using fewer terms in relation to rotation matrix (SILVA PEREIRA, 2016). In addition, quaternions solve the singularities ´problems associated to the optimization of rotation descriptions (MARINHO, 2014).

2.2 Dual Numbers

The algebra associated to the dual unit was developed by Clifford in 1871 and is known as the algebra of dual numbers, in which the dual unit ϵ is nilpotent and presents the following properties (ADORNO, 2017):

$$\epsilon \neq 0, \quad \epsilon^2 = 0 \tag{2.10}$$

A dual number is generically given by $\underline{d} = d + \epsilon d'$, wherein the coefficient d is the primary part whereas d' is the dual part. Besides, the primary and dual parts can be considered separately, applying the operators $\mathcal{P}(d)$ and $\mathcal{D}(d)$, respectively (ADORNO, 2017). Therefore:

$$\underline{d} = \mathcal{P}(\underline{d}) + \epsilon\mathcal{D}(\underline{d}) \tag{2.11}$$

The base operations of sum, subtraction and multiplication take into consideration the dual unit ϵ and are defined in(ADORNO, 2017). The dual numbers do not form a division algebra, then, the inverse can only be defined if $d \neq 0$ (ADORNO, 2017).

2.3 Dual Quaternions

Generally, the primary and dual parts of a dual number are formed by the same kind of elements, which can be scalars, complex numbers or quaternions, for instance. If the primary and dual parts are quaternions, dual numbers are customarily named dual quaternions and constitute the set $\mathcal{H}$, which presents the formal definition as (ADORNO, 2017):

$$\mathcal{H} \triangleq \{q + \epsilon q' \, : \, q,q' \, \in \, \mathbb{H} \} \tag{2.12}$$

wherein ϵ is the dual unit.

Similarly for the quaternions, the dual quaternions present analogous operations and properties (ADORNO, 2017). They have, for instance, a real and an imaginary part:

$$Re(\underline{q}) := Re(P(\underline{q})) + \epsilon Re(D(\underline{q})) \tag{2.13}$$

$$Im(\underline{q}) := Im(P(\underline{q})) + \epsilon Im(D(\underline{q})) \tag{2.14}$$

Hence, for any dual quaternion, $\underline{q} \equiv Re(\underline{q}) + Im(\underline{q})$ (ADORNO, 2017). A dual quaternion conjugate is defined as (ADORNO, 2017):

$$\underline{q}^* = Re(\underline{q}) - Im(\underline{q}) \tag{2.15}$$

and its norm is given by (ADORNO, 2017):

$$||\underline{q}|| = \sqrt{\underline{q}\,\underline{q}^*} \tag{2.16}$$

The analogous operation of vec_4 (defined for quaternions) is extended do the set $\mathcal{H}$, defined as(ADORNO, 2017):

$$vec_8(\underline{q}) = \begin{bmatrix} vec_4(P(\underline{q})) \\ vec_4(D(\underline{q})) \end{bmatrix} \tag{2.17}$$

therefore, vec_8 maps a dual quaternion into a vector (or column matrix) of real elements that represent each coordinate of the original number. There is also an analogous operation, in dual quaternions, for the hamiltonian operators, which are given by:

$$\overset{+}{H}_8(q) = \begin{bmatrix} \overset{+}{H}_4(P(\underline{q})) & 0_4 \\ \overset{+}{H}_4(D(\underline{q})) & \overset{+}{H}_4(P(\underline{q})) \end{bmatrix}, \qquad \overset{-}{H}_8(q) = \begin{bmatrix} \overset{-}{H}_4(P(\underline{q})) & 0_4 \\ \overset{-}{H}_4(D(\underline{q})) & \overset{-}{H}_4(P(\underline{q})) \end{bmatrix} \tag{2.18}$$

wherein the operators $\overset{+}{H}_8(\cdot)$ and $\overset{-}{H}_8(\cdot)$ satisfy the property:

$$\begin{aligned} vec_8(qq') &= \overset{+}{H}_8(\underline{q}) vec_8(\underline{q}') \\ &\equiv \overset{-}{H}_8(\underline{q}') vec_8(\underline{q}) \end{aligned} \tag{2.19}$$

A vector definition that can be useful is:

$$C_8 := diag(1,-1,-1,-1,1,-1,-1,-1) \tag{2.20}$$

so the following is valid:

$$vec_8(\underline{q}^*) = C_8 vec_8(\underline{q}) \tag{2.21}$$

these definitions are important to allow certain mathematical manipulations needed for the control process.

2.3.1 Rigid Motions

Rigid motions consist of the complete movement between coordinate frames (ADORNO, 2017), which is illustrated in figure 4. The use of unit dual quaternions is very convenient to represent rigid motions, and they also belong to the set (ADORNO, 2017)

$$\underline{\mathcal{S}} \overset{\triangle}{=} \{\underline{q} \in \mathcal{H} : ||\underline{q}|| = 1\} \tag{2.22}$$

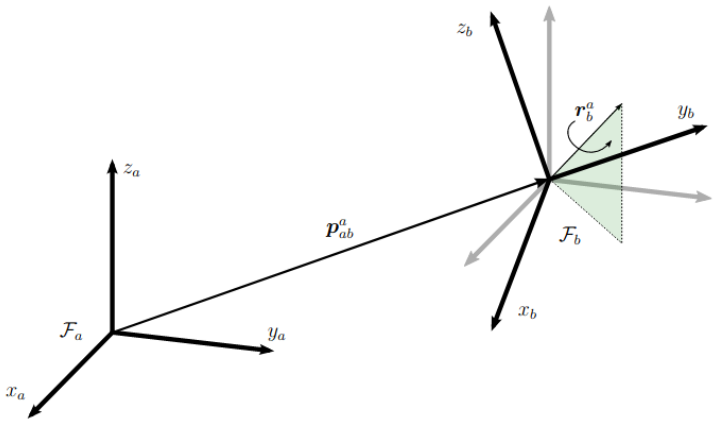

Figure 4 – Rigid motion represented by quaternions (ADORNO, 2017)

The elements of $\underline{\mathcal{S}}$, taking into account the multiplication operation, represent the ones of $Spin(3) \ltimes \mathbb{R}^3$, the group of rigid motions that double covers $SE(3)$(SELIG, 2005). Considering a quaternion $\boldsymbol{r} \in SO(3)$ and an imaginary quaternion $\boldsymbol{t}$, the unit dual quaternion associated to the translation $\boldsymbol{t}$ followed by the rotation $\boldsymbol{r}$ is determined as

$$\underline{x} = r + \frac{\epsilon}{2} t\, r \tag{2.23}$$

The concatenation of rigid transformations is calculated by the multiplication of a sequence of dual quaternions (ADORNO, 2017). Besides, for a dual quaternion $\underline{x} \in \underline{\mathcal{S}}$, the inverse transformation of rigid motion corresponds to the conjugate $\underline{x}^*$, which belongs to the group inverse of $SO(3) \ltimes \mathbb{R}^3$ because $\underline{x}^* \underline{x} = \underline{x}\, \underline{x}^* = 1$ (SELIG, 2005).

2.4 Kinematic Control

As for the mathematical description of a robot manipulator kinematics, it is expressed the vector x related to pose of the end-effector, which has eight elements (SILVA PEREIRA, 2016):

$$x = vec_8(\underline{x}) \tag{2.24}$$

wherein $\underline{x}$ is the dual quaternion of the manipulator's pose. The vector θ, that contains the joint variables $\theta_1, \theta_2, ..., \theta_n$, may be expressed as a function of x, which is a function related to the Inverse Kinematics (IK) (SILVA PEREIRA, 2016):

$$\theta = g(x) \tag{2.25}$$

To determine, analytically, this function $g(x)$ can be extremely difficult, even not possible in some cases, having to resort to numerical algorithms. However, the proposal of this work involves another mathematical artifice to work around this problem, using analytic expressions. First, it is necessary to express the function of direct kinematics (SILVA

PEREIRA, 2016):

$$x = f(\theta) \tag{2.26}$$

The function $f(\theta)$ is usually obtained by the Denavit-Hartenberg parameters. Differentiating equation 2.26 with respect to time leads to

$$\dot{x}(t) = \frac{\partial f(\theta)}{\partial \theta}\frac{\partial \theta}{\partial t} \tag{2.27}$$

wherein

$$\frac{\partial f(\theta)}{\partial \theta} := J = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial \theta_1} & \cdots & \frac{\partial f_8}{\partial \theta_n} \end{bmatrix} \tag{2.28}$$

in which J is known as the Jacobian matrix of the manipulator.

Therefore, the so called differential kinematics can be solved by minimizing the cost function

$$J(q)\dot{\theta}(t) - \dot{x}_d(t) \tag{2.29}$$

wherein $x_d(t) = vec_8(\underline{x}_d)$ is the desired pose vector, with $\underline{x}_d$ being the dual quaternion of the end-effector. To achieve an exponential decay, a closed loop with error $e_c = x_d - x$ feedback can be constructed. Hence, the cost function can actually be defined as (SILVA PEREIRA, 2016)

$$J\dot{\theta} - \dot{x}_d + K(x - x_d) = 0 \tag{2.30}$$

in which K is a positive real number.

2.4.1 Proportional Controller for Invariant Error Function

The error function, instead of being expressed as $x_d - x$ (as in equation 2.30), can be be defined as (SILVA PEREIRA, 2016):

$$\underline{e}_c = 1 - \underline{x}^*\underline{x}_d \tag{2.31}$$

in which $\underline{e}_c$ is the invariant error function. This name is justified by the fact that this error presents the same value for different reference systems for the pose (SILVA PEREIRA, 2016). The time derivative of $\underline{e}_c$ is given by (SILVA PEREIRA, 2016):

$$\dot{\underline{e}}_c = -\dot{\underline{x}}^*\underline{x}_d - \underline{x}^*\dot{\underline{x}}_d \tag{2.32}$$

therefore:

$$\begin{aligned} vec_8(\dot{\underline{e}}_c) &= vec_8(-\dot{\underline{x}}^*\underline{x}_d) - vec_8(\underline{x}^*\dot{\underline{x}}_d) \\ &= -\bar{H}(\underline{x}_d)vec_8(\dot{\underline{x}}^*) - vec_8(\underline{x}^*\dot{\underline{x}}_d) \\ &= -\bar{H}(\underline{x}_d)C_8J\dot{\theta} - vec_8(\underline{x}^*\dot{\underline{x}}_d) \\ &\equiv -N\dot{\theta} - vec_8(\underline{x}^*\dot{\underline{x}}_d) \end{aligned} \tag{2.33}$$

wherein $N := \bar{H}(\underline{x}_d)C_8J$. In order to produce an exponential decay in the invariant error function, it is imposed that $\dot{\underline{e}}_c = -K\underline{e}_c$ (again, K is a positive real gain). Hence, the control law is determined by:

$$\dot{\theta} = N^{\dagger}(K\, vec_8(\underline{e}_c) - vec_8(\underline{x}^*\dot{\underline{x}}_d)) \tag{2.34}$$

in which $N^{\dagger}$ is the Moore Penrose pseudo-inverse of N.

3 Model Predictive Control

3.1 Introduction

In the mid 70's, the Model Based Predictive Control (MBPC), or simply Model Predictive Control (MPC) emerged and has noticeably evolved thenceforth (CAMACHO; BORDONS; ALBA, 2004). This paradigm isn't considered a specific control approach, it actually encompasses a wide variety of control strategies (CAMACHO; BORDONS; ALBA, 2004). Linear multi-variable controllers can then be obtained by the use of MPC, considering a discrete or continuous system (WANG, 2009). For a control strategy to be considered MPC, some directives must be fulfilled (CAMACHO; BORDONS; ALBA, 2004): expressly acknowledge a mathematical model for the process in order to predict the future outputs of the system, within a finite time horizon; a sequence of control signals must be computed in pursuance of the minimization of an objective function; the idea of a receding predictive horizon, which consists of a fixed time interval, counting from the current instant, must be examined in order to make the outputs and inputs prediction.

Figure 5 shows an example of time functions for the control input $u(t)$ and the controlled output $y(t)$. With the help of this illustration, the MPC methodology can be explained. Firstly, it is important to highlight that the notation $(t+k|k)$ indicates that the quantity is considered at the instant $t+k$ but was computed in the instant t, wherein k varies from 1 to N. For a specified prediction horizon N, the future outputs $y(t+k|k)$ rely upon the current control signal and output variables and also the predicted inputs $u(t+k|k)$ (CAMACHO; BORDONS; ALBA, 2004).

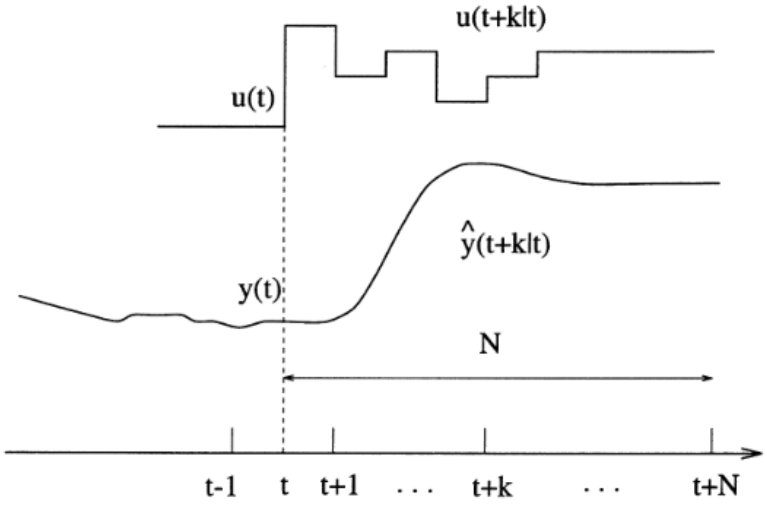

Figure 5 – Illustration of the input and output in a MPC system (CAMACHO; BORDONS; ALBA, 2004)

The control law for a MPC is computed by the optimization of some mathematical formulation that tends to lead the outputs to follow a reference signal $r(t+k)$ (CAMACHO; BORDONS; ALBA, 2004). Mainly, this design can be formulated as a quadratic cost function of the error computed by the difference of the predicted output variables in relation to the predicted reference signal (CAMACHO; BORDONS; ALBA, 2004). The solution will then generate a set of control inputs, in which $u(t|t)$ is the one to be applied in the system (CAMACHO; BORDONS; ALBA, 2004). The future control signals are discarded, since, in the later iterations, the values needed to compute them might have changed.

3.2 MPC applied to manipulators

The use of redundant manipulators is very recurrent in industry and in research, due to its versatility in the task execution (FARONI; BESCHI; PEDROCCHI, et al., 2019). Some functions, like manipulability maximisation, joint availability or minimum energy consumption, may be completely or partially satisfied, using the additional degrees of freedom (FARONI; BESCHI; TOSATTI, 2017).

It is important to accomplish a task in a way that it does not interfere in the performance of eventual tasks with higher priority. In (FARONI; BESCHI; PEDROCCHI, et al., 2019), it is made a review of some methods that manage secondary tasks while the execution of primary one is not affected. These approaches, however, do not manifestly consider the existence of boundaries for the manipulator's movement, which may lead to a inverse kinematic solution that is inconvenient or even impracticable for the manipulator. It is also quoted in (FARONI; BESCHI; PEDROCCHI, et al., 2019) methodologies that take into account rigid limits of redundant robot manipulators, creating a cost function which induces the joint positions to remain within the pre-established limits. It is also mentioned in (FARONI; BESCHI; PEDROCCHI, et al., 2019) a reference that handles kinematic bounds, such as joint velocities and accelerations, through a procedure that demands the computation of a weighted pseudo-inverse matrix related to the constrained inverse kinematics. The inconvenience of these approaches is that the robot limits are not necessarily respected, even in the quoted procedures that take them into account. Therefore, the actuators might get saturated or even damaged, culminating in the unfulfillment of the desired trajectory.

To handle the physical limits of the manipulator, a relevant trend lately has been the formulation of the inverse kinematic as a Least Square Problem (LSP) (FARONI; BESCHI; PEDROCCHI, et al., 2019). It can be made a search in a set of joint variables that satisfy both the trajectory and the robot constraints, while eventual tasks can be managed. The whole point of converting the IK problem into a LSP consists in the fact that it can be effectively solved, including hard bounds, by numerical quadratic programming (QP) solvers (FARONI; BESCHI; PEDROCCHI, et al., 2019). QP consists in the optimization of an objective function

that is quadratic in regard of some set of variables; this method is considered to be one of the most convenient for non-linear programming (FRANK; WOLFE, 1956). The possibilities are that from bi-linear to second degree polynomial terms might be inserted in the objective function, whereas the constraints have to be linear and might be expressed by inequalities or equalities (FLOUDAS; VISWESWARAN, 1995).

The constraints considered for the manipulator influence the quality of fulfillment of the assignment. Usually, the primary task is defined as a trajectory. A very comprehensive procedure to keep a robot on its geometrical path is the time scaling of the task, which permits the trajectory to be made in a shorter period. There are some methods quoted in (FARONI; BESCHI; PEDROCCHI, et al., 2019) that use time scaling, through different approaches.

The previous works mentioned in (FARONI; BESCHI; PEDROCCHI, et al., 2019) consider, in each iteration for the respective algorithm used, information about only the current configuration of the robot. They are, hence, known as local methods and present a solution that is optimal in relation to the current state, but not necessarily in relation to the total accomplishment of the task. As demonstrated in (FARONI; BESCHI; TOSATTI, 2017), the solution of a local optimization problem might lead to a result that is unfavorable for the global execution of the assignment, if compared to an algorithm that takes into account the task evolution. The high computational cost is a considerable limitation for calculating the solution for the IK and the scaling factor considering the global task. Therefore, as in (FARONI; BESCHI; PEDROCCHI, et al., 2019), in this project, it is used the paradigm of MPC with a finite predictive horizon to compute an optimal control law.

3.3 Mathematical formulations

For an initial development, it is considered the following equation for the inverse kinematics:

$$s\dot{x}(\sigma) = J(q)\dot{\theta}(t) \tag{3.1}$$

wherein the term s is the scaling element, enabling the time dilation of the task, as long as it is imposed that its value remains between one and zero (FARONI; BESCHI; PEDROCCHI, et al., 2019). As for σ, it corresponds to the dilated time function defined for the pose task x, wherein:

$$\sigma(t) = \int_0^t s \, d\tau \tag{3.2}$$

therefore, $s = d\sigma/dt$. Considering the discrete-time system for implementation in a simulation environment, the following approximation is adopted:

$$\sigma[k] = \sigma[k-1] + Ts[k-1] \tag{3.3}$$

in which $\sigma[0] = 0$, $s[0] = 1$ and T corresponds to the sampling time of the control system involved. Equation 3.3 is the one used to compute the parameter σ used in each iteration for the assigned task $x(\sigma)$.

Considering the optimization problem shown in equation 2.30, the inverse kinematics problem with proportional error feedback adapted with the inclusion of a scaling factor can be given by

$$J\dot{\theta} - s(\dot{x}_d - K(x - x_d)) = 0 \tag{3.4}$$

This formulation in equation 3.4 can be converted into a LSP (enabling to find a solution by a QP solver) such as:

$$\Omega_1 := ||J\dot{\theta} - s(\dot{x}_d - K(x - x_d))||^2 \tag{3.5}$$

$$\Omega_2 := (1 - s)^2 \tag{3.6}$$

in which Ω_1 is a cost function with higher priority then the cost function Ω_2. To solve both the cost functions and, eventually, other ones with lower priorities can be challenging, demanding complex methods of programming, specially if the hard bounds are considered. A more straightforward procedure is to convert Ω_1 and Ω_2 into:

$$\Omega := \lambda_1||J\dot{\theta} - s(\dot{x}_d - K(x - x_d))||^2 + \lambda_2(1 - s)^2 \tag{3.7}$$

wherein λ_1 and λ_2 are the Lagrange multipliers. If it is established a much larger value for λ_1 than for λ_2, the cost function Ω can fulfill the role of considering the function Ω_1 as being the primary one to be minimized and, secondly, minimize Ω_2 (FARONI; BESCHI; PEDROCCHI, et al., 2019). The hard bounds can be expressed by:

$$\Theta_{min} \leq \theta \leq \Theta_{max} \tag{3.8}$$

$$\dot{\Theta}_{min} \leq \dot{\theta} \leq \dot{\Theta}_{max} \tag{3.9}$$

$$\ddot{\Theta}_{min} \leq \ddot{\theta} \leq \ddot{\Theta}_{max} \tag{3.10}$$

$$0 \leq s \leq 1 \tag{3.11}$$

However, the minimization of the cost function Ω produces results only for s and for $\dot{\theta}$. Hence, the constraints for the joint positions and accelerations must be formulated as a function of the joint velocities. For a discrete system, this bounds can be approximated as:

$$\Theta_{min} \leq \theta(t) + T\dot{\theta}(t) \leq \Theta_{max} \Rightarrow \frac{\Theta_{min} - \theta(t)}{T} \leq \dot{\theta}(t) \leq \frac{\Theta_{max} - \theta(t)}{T} \tag{3.12}$$

$$\ddot{\Theta}_{min} \leq \frac{\dot{\theta}(t) - \dot{\theta}(t - T)}{T} \leq \ddot{\Theta}_{max} \Rightarrow T\ddot{\Theta}_{min} + \dot{\theta}(t - T) \leq \dot{\theta}(t) \leq T\ddot{\Theta}_{max} + \dot{\theta}(t - T) \tag{3.13}$$

Finally, the problem of calculating the scaling factor and the joint velocities is formally defined as:

$$\begin{bmatrix} \dot{\theta} \\ s \end{bmatrix} = \underset{\begin{bmatrix} \dot{\theta} & s \end{bmatrix}^T \in \zeta}{argmin} (\Omega) \tag{3.14}$$

wherein ζ corresponds to the set that limits $\dot{\theta}$ and s to their pre-defined restraints. A numerical QP solver can determine the optimal solution.

As for the global method, which applies the paradigm of MPC, needs to take into account, as already mentioned, the progression of the variables along the predictive horizon. The primary idea of the procedure is to develop, at each time step, the resolution of the predictive IK cost function with the scaling factor as a constrained QP. Thereupon, in the interest of receding horizon control, the first input of the sequence is utilized in the system. This operation is remade until all the path is covered.

3.3.1 Conception of the Predictive Model

The kinematic predictive model for a robot manipulator can be obtained by formulating a state-space description, as developed in (FARONI; BESCHI; TOSATTI, 2017). Even if the implementation model is discrete, the work in (FARONI; BESCHI; TOSATTI, 2017) considered a continuous time function for the kinematic variables, what enabled an analysis that was convenient for a relatively long prediction horizon. Taking this into account and the small homogeneous sampling time period, it would lead to a deeply extensive quantity of control variables taking part in the predictive model (DIMITROV et al., 2008).

In the interest of reducing the intricacy of the predictive model, the methodology proposed in (FARONI; BESCHI; TOSATTI, 2017) has been utilized. Such strategy enables to have a small quantity of prediction and control time instants along the prediction horizon, instead of considering all the sampling times. Expressly, for a sequence of prediction time instants $\{\tau_i\}, 1 \leq i \leq p$, and a sequence of control time instants $\{\bar{t}_j\}, 1 \leq j \leq c$, the predictive horizon (given in seconds) is denoted by τ_p . The control horizon (also in seconds) is named $\bar{t}_c$.

3.3.2 Selection Criteria of the Predictive and Control Instants

The predictive formulation depends on the pre-established distribution of the prediction and control time instants τ_i and $\bar{t}_j$ throughout the respective predictive horizons. Despite the fact that prediction and control time instants don't need to be imposed as equal, henceforward they are chosen to be the same, which does not imply in loss of generality. Considering that the reference signal (corresponding to the geometrical assignment) is not typically constant, an optimal distribution for τ_i and $\bar{t}_j$ cannot be determined. Thereupon, it is adopted, similar to (FARONI; BESCHI; TOSATTI, 2017), an elementary empiric tuning rule established on the resulting directives:

- The control and prediction time instants are equally distributed: $\tau_i = \bar{t}_i$.
- The first time instant is equal to zero: $\tau_1 = \bar{t}_1 = 0$.

- The difference between two considered time instants grows as the index increases.

Therefore, for a specific quantity of time instants $c = p$ and the width of the horizon τ_p given in seconds, (FARONI; BESCHI; PEDROCCHI, et al., 2019) proposes a parabolic distribution of τ_i and $\bar{t}_i$ in the form:

$$\tau_i = \bar{t}_i = \frac{\tau_p}{(p-1)^2}(i-1)^2 \tag{3.15}$$

in which $1 \leq i \leq p$. And so, a relatively small amount of time instants is enough to work with both the predictive and control time horizons.

3.3.3 Task Fulfillment

The fulfillment of the primary task by the means of the global method relies also upon the resolution, through a QP solver, of the IK problem formulated as a LSP, which, similar to (FARONI; BESCHI; PEDROCCHI, et al., 2019), considers the time instants τ_i mentioned in the section 3.3.2. The cost function of IK, formulated as a LSP, is given by:

$$\Omega_1 := || J^*(\Theta_{(\tau)})\dot{\Theta}_{(\tau)} - (\dot{X}_{d(\tau)} - K(X_{(\tau)} - X_{d(\tau)}))S(\tau) ||^2 \tag{3.16}$$

wherein

$$\begin{aligned}
&J^*(\,\Theta(\tau)\,) = \text{blkdiag}(J(\,\theta(t+\tau_1)\,),\,\dots,\,J(\,\theta(t+\tau_p)\,)\,)\\
&\Theta(\tau) = [\;\theta(t+\tau_1)^T\,,\dots,\theta(t+\tau_p)^T\;]^T\\
&S(\tau) = [\;s(t+\tau_1)^T\,,\dots,s(t+\tau_p)^T\;]^T\\
&X(\tau) = \text{blkdiag}(\,x(\,\sigma(t)+\tau_1\,),\,\dots,\,x(\,\sigma(t)+\tau_p\,)\,)\\
&X_d(\tau) = \text{blkdiag}(\,x_d(\,\sigma(t)+\tau_1\,),\,\dots,\,x_d(\,\sigma(t)+\tau_p\,)\,)
\end{aligned}$$

in which *blkdiag* corresponds to a kind of matrix that is formed as a function of other smaller matrices that fill the diagonal of the block diagonal matrix. This type of matrix are important to enable the calculations needed to determine the solutions for the LSP. This matrices in equation 3.16 are augmented versions of the ones in equation 3.4.

Strictly analyzing, equation 3.16 is non-linear in relation to the control signal $\dot{\Theta}$, as the augmented Jacobian matrix is a function of current and future joint positions (that relies upon the current and future joint velocities). Hence, an approximation of the problem is considered, so it can be actually considered as a LSP: the future joint positions used in the Jacobians are calculated by a simple integration of the joint velocities computed in the previous iteration (considering them as linear functions of time). For the first loop, the joint velocities used for the calculation of the approximated Jacobians are zero.

The vector containing the scaling factors must also be computed by the QP solver. Analogously to the local method, the optimization regarding the scaling factor has lower

priority than the trajectory task. The cost function to work with the scaling factor is given by:

$$\Omega_2 := ||1_{p\times 1} - s_{(\tau)}||^2 \tag{3.17}$$

in which $1_{p\times 1}$ is a column matrix formed by ones in all its p lines.

Just like for the local method, the priority of the cost functions Ω_1 and Ω_2 can be formulated into one single cost function Ω as:

$$\Omega := \lambda_1 || J^*(\Theta_{(\tau)})\dot{\Theta}_{(\tau)} - (\dot{X}_{d(\tau)} - K(X_{(\tau)} - X_{d(\tau)}))s(\tau) ||^2 + \lambda_2||1_{p\times 1} - s_{(\tau)}||^2 \tag{3.18}$$

wherein λ_1 needs to have a much larger value than λ_2 to emulate the difference of priorities. Also for this method, the trajectory task is the primary one, while the maximization of the scaling factor is the secondary.

The hard bounds considered for the local method are also taken into account in the global one. However, their formulation has a subtle difference:

$$\begin{aligned} &\dot{\Theta}_{min} \leq \theta(t+\tau_i) + T\dot{\theta}(t+\tau_i) \leq \dot{\Theta}_{max} \\ &\Rightarrow \frac{\Theta_{min} - \theta(t+\tau_i)}{T} \leq \dot{\theta}(t+\tau_i) \leq \frac{\Theta_{max} - \theta(t+\tau_i)}{T} \end{aligned} \tag{3.19}$$

and:

$$\begin{aligned} &\ddot{\Theta}_{min} \leq \frac{\dot{\theta}(t+\tau_i) - \dot{\theta}(t+\tau_i - T)}{T} \leq \ddot{\Theta}_{max} \\ &\Rightarrow T\ddot{\Theta}_{min} + \dot{\theta}(t+\tau_i - T) \leq \dot{\theta}(t+\tau_i) \leq T\ddot{\Theta}_{max} + \dot{\theta}(t+\tau_i - T) \end{aligned} \tag{3.20}$$

the inequalities are applied to the different time instants considered in the predictive horizon. Therefore, there are constraints between the control signals for each instant τ_i.

Hence, the problem of calculating the predictive scaling factor and joint velocities is formally defined as:

$$\begin{bmatrix} \dot{\Theta} \\ S \end{bmatrix} = \underset{\left[\dot{\Theta} \quad S\right]^T \in \zeta}{argmin\ (\Omega)} \tag{3.21}$$

wherein ζ corresponds to the set that limits $\dot{\Theta}$ and S to their pre-defined restraints. The same numerical QP solver used for the local method can compute the optimal solution for the MPC method.

4 Development and Implementation

4.1 CoppeliaSim

Robotic systems integrates the use of actuation, sensing and control, which makes them powerful but also complex. Such systems depend upon essential apparatus and a consistent background on kinematics, dynamics, motion planning, control techniques and, eventually, computer vision (SPONG; HUTCHINSON; VIDYASAGAR, 2005). In general, robot manipulators employment framework needs to provide means to deal with these cited fields (SILVA PEREIRA, 2016). Hence, a multi-functional robot simulator is required to enable the simulation to work properly and be useful for different analysis.

Coppelia Robotics [1] created a robot simulator named CoppeliaSim (formerly V-REP). This software is versatile, scalable, yet mighty work environment that aims to merge all the required computational resources for complex, comprehensive simulation scenarios, such as distributed, cascade control architecture (ROHMER; SINGH, S.; FREESE, 2013).

There is a plenty of robot simulation platforms that, although provide useful functionalities, often fail to offer a wide repertoire of programming techniques, in addition to the simulation models and controllers being only partially portable: each one needs different management (ROHMER; SINGH, S.; FREESE, 2013). These restraints makes it difficult to port between hardware or platforms.

CoppeliaSim enables the choice among diversified programming procedures simultaneously. It is even possible to combine these different techniques (ROHMER; SINGH, S.; FREESE, 2013): embedded scripts in Lua, plug-ins to interface to specific hardware, remote API via socket communication, add-ons for customization and ROS. In particular, in this work, it is used a ROS node in Matlab to communicate with another ROS node in the CoppeliaSim scene containing the robot manipulator.

For the current work, CoppeliaSim was used to test the kinematic controllers in the robot manipulator Kuka LBR4p shown in figure 6. It is important to simulate different trajectories, under different constraints, before implementing the controllers in the actual robot. It permits to verify if there are stability issues in controllers, caused by possible numerical conditioning, numerical drifting and other eventual problems. Therefore, the simulation can prevent possible damages to the actual manipulator.

The software used provide different kinds of elemental objects (ROHMER; SINGH, S.; FREESE, 2013):

[1] https://www.coppeliarobotics.com

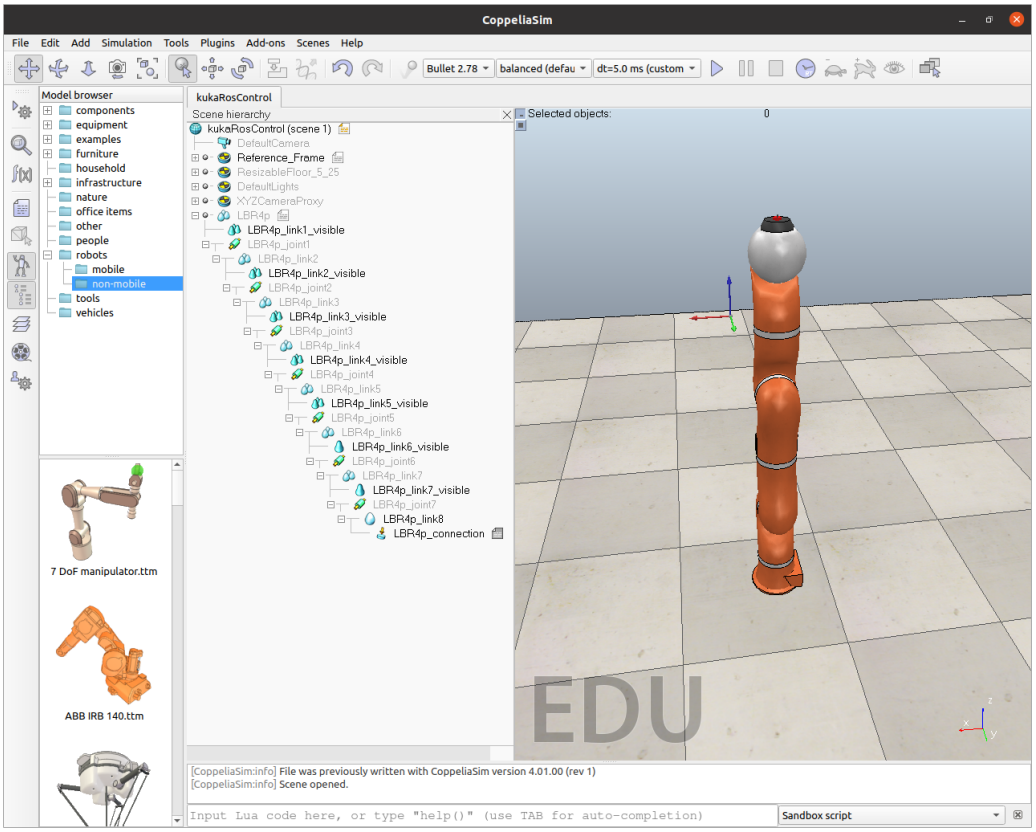

Figure 6 – Kuka LBR4p used for the simulations in CoppeliaSim

- **Joints**: feature that connects two or more scene objects, presenting one to three degrees of freedom, depending if the joint is prismatic, revolute, screw-like or spherical. They can work in distinct modes, like in force, torque or inverse kinematics mode. The interesting thing is that each joint can be associated with a script, both to apply a control law, and to publish data, such as effort, speed and position.
- **Shapes**: triangular meshes, with usage in rigid body simulation and visualization. The dynamics module depend greatly on shapes for its calculations.
- **Graphs**: element that enables to record and directly display data, in different ways.
- **Cameras**: make it possible to visualize a scene with a specific view-port.
- **Lights**: they can illuminate the whole scene or specific scene objects, which influences the view through cameras.
- **Dummies**: a reference frame that can be placed in a scene, which helps in the fulfillment of multiple tasks.

In figure 6, it is possible to observe, in a list in the left part of the screen, the existence of seven joints (LBR4p_joint1 to LBR4p_joint7), eight links with dynamics module enabled (LBR4p_link1 to LBR4p_link8) and seven visible links (LBR4p_link1_visible to

LBR4p_link7_ visible) that enables the robot to be illustrated. The elements cited are structured as a hierarchy tree, wherein each joint's movement influences in the links that are listed below the respective joint.

4.2 Robot Operating System (ROS)

The integration of diverse systems can be a critical aspect to achieve good results in controlling experimentation. There is a plenty of communication protocols and computer architectures that can enable to exchange data between different programs. Complex robotic systems require such tasks, as they are continually in a scale growth. A suitable framework is Robotic Operating System (ROS) [2], as it allows to write robot software in a flexible way. ROS is a meta-operating, open-source system dedicated for robot programming (YOONSEOK PYO HANCHEOL CHO; LIM, 2017), designed with the objective of associating large-scale robotics applications (QUIGLEY et al., 2009). This tool implements typical features of an operating system, such as data transfer between processes and package handling (YOONSEOK PYO HANCHEOL CHO; LIM, 2017).

ROS was used in this work to communicate the robot simulation in CoppeliaSim with a program in Matlab that computes the task planning and the control law. The main ROS features utilized were ROS nodes [3] and topics [4]. A node corresponds to a program that can be executed to realize computations needed for a process that can be visible, communicable via ROS (YOONSEOK PYO HANCHEOL CHO; LIM, 2017). Nodes can be connected in a graph and exchange data through ROS topics and are supposed to communicate at a large rate (YOONSEOK PYO HANCHEOL CHO; LIM, 2017). All the nodes being executed present a unique identification in the graph that differs each one within the system (YOONSEOK PYO HANCHEOL CHO; LIM, 2017).

The employment of ROS nodes present a series of advantages (YOONSEOK PYO HANCHEOL CHO; LIM, 2017):

- Tolerance to failures is higher, since breakdowns are segregated into separate nodes;
- In relation to monolithic systems, the complexity of algorithms is diminished;
- The nodes advertise the least possible API to the other part of the graph, which enables to omit particular technicalities of the implementation;
- Distinct nodes can be implemented in different programming languages and still have an efficient communication.

[2] http://www.ros.org
[3] http://wiki.ros.org/Nodes
[4] http://wiki.ros.org/Topics

Topics consist in buses through which nodes transfer and receive data and present particular publishing and subscribing semantics, which decouples the data sources from its acquisition (YOONSEOK PYO HANCHEOL CHO; LIM, 2017). Usually, it is not transparent to each node what are the other parts in the communication process. Actually, nodes can subscribe to a topic to receive a certain type of messages or publish data to the topic involved (YOONSEOK PYO HANCHEOL CHO; LIM, 2017).

In this project, there are two nodes for each kinematic controller: the controlling node in Matlab, that performs calculations in dual quaternions needed to obtain the control law; the node in the simulator CoppeliaSim, that provides the sensing necessary information about the poses and joint variables for the control module.

4.3 DQ Robotics

There are computational frameworks that are more suitable to work with complex calculations, like the ones needed in the process of robot modelling and kinematic control using the set of dual quaternions. Fortunately, to this purpose, there is a library provided for Matlab(as for Python and C++) named DQ Robotics (ADORNO; MARINHO, 2021), which is a very convenient to understand and use, besides being computationally efficient. Reference (ADORNO; MARINHO, 2021) mentions some libraries that provide tools to work with quaternions and dual quaternions in Lua, MATLAB and C++, but not with functionalities suitable for robot manipulators as DQ Robotics.

DQ Robotics supports object-oriented commands. There is a class named DQ that encompasses the main methods of this library are shown in figure 7. These methods are very important to compute the necessary calculations involving dual quaternions, such as conversions between coordinate systems and other operations related to determine the control law.

In figure 8, it is illustrated a simplified UML class diagram of the robot modeling classes with the main methods available. Essentially, this project has used the classes DQ_Kinematics and DQ_SerialManipulator (which inherits the methods of the former class). The constructor method of DQ_SerialManipulator requires the Denavit-Hartenberg parameters, which enables to compute direct kinematics and the pose jacobian, for instance.

4.4 Methodology

The controllers implemented in this project were developed in Matlab, since this program has support to communicate through ROS, supports the library DQ robotics and posses an efficient QP solver, the function **quadprog**, which can consider equality and inequality constraints in the interest of minimizing a cost function. In the case of this

Binary operations between two dual quaternions (e.g., a + b)	
+, -, *	Sum, subtraction, and multiplication.
/, \	Right division and left division.
Unary operations (e.g., -a and a')	
-	Minus.
['], [.']	Conjugate and sharp conjugate.
Unary operators (e.g., P(a))	
conj, sharp	Conjugate and sharp conjugate.
exp, log	Exponential of pure dual quaternions and logarithm of unit dual quaternions.
inv	Inverse under multiplication.
haminus4, haminus4	Hamilton operators of a quaternion.
haminus8, haminus8	Hamilton operators of a dual quaternion.
norm	Norm of dual quaternions.
P, D	Primary part and dual part.
Re, Im	Real component and imaginary components.
rotation	Rotation component of a unit dual quaternion.
rotation_axis, rotation_angle	Rotation axis and rotation angle of a unit dual quaternion.
vec3, vec6	Three-dimensional and six-dimensional vectors composed of the coefficients of the imaginary part of a quaternion and a dual quaternion, respectively.
vec4, vec8	Four-dimensional and eight-dimensional vectors composed of the coefficients of a quaternion and a dual quaternion, respectively.
Binary operators (e.g., Ad(a,b))	
Ad, Adsharp	Adjoint operation and sharp adjoint.
cross, dot	Cross product and dot product.
Binary relations (e.g. a == b)	
==, ~=	Equal and not equal.
Common methods	
[plot]	Plots coordinate systems, lines, and planes.

Methods and operations enclosed by brackets (e.g., [.]) are available only on MATLAB.

Figure 7 – Main methods of Class DQ (figure extracted from (ADORNO; MARINHO, 2021))

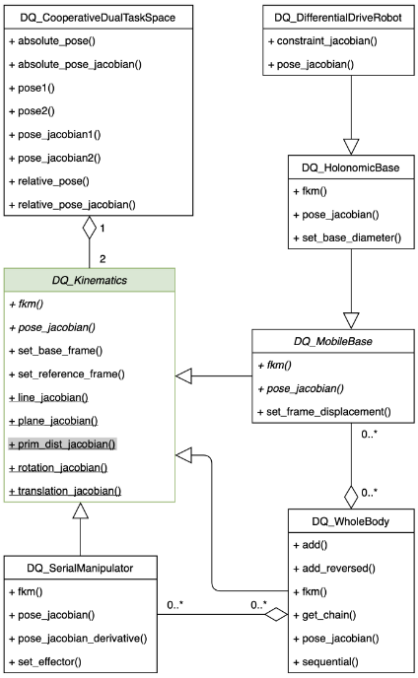

Figure 8 – Simplified UML class diagram provided by DQ Robotics (ADORNO; MARINHO, 2021)

work, the cost function is the kinematic equation including the feed-forward term, as in equation 2.30 for the methods that use the QP solver. As for the proportional controller, the control law is obtained through equation 2.34.

The controlled trajectory is that of the end-effector. The measurements that enable to calculate its pose is the joint angles of the Kuka manipulator. However, this computed pose is in relation to the robot's base. The measured poses, though, are provided from the CoppeliaSim to the Matlab with respect to the world coordinate frame of the simulator. A pose x defined with respect to the reference frame shown in figure 6 (the one floating near

to the robot) can be expressed, in relation to the coordinate frame attached to the robot's base, as

$$\underline{x} = \underline{b}_p^* \underline{\underline{ref}}_p \underline{x}_0 \tag{4.1}$$

wherein $\underline{b}_p$ is the robot's base pose (in relation to the world coordinate frame), $\underline{\underline{ref}}_p$ is the reference frame pose (in relation to the world coordinate frame) and $\underline{x}_0$ is a generic pose defined with respect to reference frame. Therefore, a desired trajectory can be specified in relation to this reference frame, which is the procedure done in this project.

4.4.1 The Implementation

The first procedure to start working with the robot is to determine its Denavit-Hartenberg parameters. The robot manipulator Kuka LBR4p has seven degrees of freedom, in which all the joints are rotational. Hence, the variable parameters of Kuka's DH matrix are the angles θ (the joint angular positions), which enable to compute the direct kinematics. The DH matrix of Kuka is determined as

$$DH_{Kuka} = \begin{bmatrix} DH_\theta \\ DH_d \\ DH_a \\ DH_\alpha \end{bmatrix} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 & \theta_7 \\ 0.2 & 0 & 0.4 & 0 & 0.39 & 0 & 0.078 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \pi/2 & -\pi/2 & \pi/2 & -\pi/2 & \pi/2 & -\pi/2 & 0 \end{bmatrix} \tag{4.2}$$

which was constructed from measurements and observing the orientation of the coordinate frames in a scene with the robot in CoppeliaSim. In possession of DH_{Kuka}, it is necessary to measure the pose of the first link of the manipulator with respect to its base, so the direct kinematics can be calculated correctly. This pose of the first link is given by

$$\underline{l}_p = 1 + 0.5\epsilon(0.51\hat{k}) \tag{4.3}$$

as the link rotation is null and it is dislocated $0.51m$ from the base (which was also verified using the scene in CoppeliaSim).

Afterwards, the rate of communication by ROS between CoppeliaSim and the controlling node in Matlab is set as $T = 0.05s$. The programs in Matlab and in CoppeliaSim determine its publishers and subscribers: the first one sends the desired joint velocities to achieve a certain pose by the end-effector while the simulator sends the joint state (containing the measured angular positions, velocities and effort in each joint), but firstly the program in CoppeliaSim provides the poses $\underline{\underline{ref}}_p$ and $\underline{b}_p$ needed by the controlling node in Matlab.

4.4.1.1 Solutions without time scaling

For the proportional controller using the invariant error function, the control loop must simply compute the control law expressed in equation 2.34. For the MPC method,

before beginning the loop, the controlling node must define the constraints and some other parameters to provide for the QP solver. The function **quadprog** used demands matrices as arguments to compute the control action $u(t)$, which are named H, f, A, b, lb and ub. The **quadprog** function works to minimize the cost function:

$$\frac{1}{2}u^T H u + f^T u \tag{4.4}$$

restrained to the inequalities

$$\begin{cases} Au \leq b \\ lb \leq u \leq ub \end{cases} \tag{4.5}$$

In terms of the kinematic variables, the cost function is

$$C_\omega := ||J\dot{\theta} - \dot{x}_d + K(x - x_d)||^2 \tag{4.6}$$

Considering $\dot{x}_d = vec_8(\underline{\dot{x}}_d)$, $x_d = vec_8(\underline{x}_d)$ and $x = vec_8(\underline{x})$, the cost function becomes

$$C_\omega := \dot{\theta}J^T J\dot{\theta} - 2(K(X_d - x) - \dot{x}_d)^T J + \phi \tag{4.7}$$

wherein ϕ is a number that depends on the other kinematic variables, but not on the the control signals. Therefore:

$$\begin{aligned} H &= J^T J \\ f &= -J^T(K(x_d - x) + \dot{x}_d) \end{aligned} \tag{4.8}$$

4.4.1.2 Solutions with time scaling

In terms of the kinematic variables, the cost function is

$$C_\omega := \lambda_1||J(\theta)\dot{\theta} - (\, vec_8(\underline{\dot{x}}_d) + K\, vec_8(\underline{x}_d - \underline{x})\,)s||^2 + \lambda_2||1 - s||^2 \tag{4.9}$$

Considering $\dot{x}_d = vec_8(\underline{\dot{x}}_d)$, $x_d = vec_8(\underline{x}_d)$ and $x = vec_8(\underline{x})$, the cost function becomes

$$C_\omega = \lambda_1(\, J(\theta)\dot{\theta} - (\, \dot{x}_d + K(x_d - x)\,)s\,)^T(\, J(\theta)\dot{\theta} - (\, \dot{x}_d + K(x_d - x)\,)s\,) + \lambda_2(1-s)^T(1-s) \tag{4.10}$$

Developing equation 4.10 into a matrix form, it is possible to conclude:

$$H = \begin{bmatrix} \lambda_1 J^T J & -\lambda_1 J^T(Ke_c + \dot{x}_d) \\ -\lambda_1(Ke_c + \dot{x}_d)^T J & \lambda_1(Ke_c + \dot{x}_d)^T(Ke_c + \dot{x}_d) + \lambda_2 I \end{bmatrix} \tag{4.11}$$

$$f^T = \begin{bmatrix} 0_{n_j p} & -\lambda_2 1_p \end{bmatrix} \tag{4.12}$$

wherein $e_c = x_d - x$ is the control error function and I is the identity matrix. For the Local method, $p = 1$ while for the MPC method, it was tested $p = 6$ and $p = 10$. The dimensions of the matrices considered rely upon the value of p, as exposed in section 3.3.3.

4.4.1.3 Definition of the constraints

As for the restrictions it was considered joint ranges to be:

$$-\pi \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2.5 \\ 2.5 \\ 2.5 \end{bmatrix} \leq \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \\ \dot{\theta}_7 \end{bmatrix} \leq \pi \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2.5 \\ 2.5 \\ 2.5 \end{bmatrix} \tag{4.13}$$

which were empirically obtained values that worked well for the situations tested.

To work with the feasibility of the trajectory, it was considered a restraint of mean acceleration, that was estimated, for the circle trajectory, considering the measured torques provided by the ROS node in CoppeliaSim and the measured velocities as

$$\tau_{max} = In|\alpha_{mean}| = In\frac{\left|\dot{\theta}_{(n-1)} - \dot{\theta}_{(n-2)}\right|}{\Delta t} \tag{4.14}$$

wherein $\dot{\theta}_{k-l}$ is the measured vector of joint velocities in the iteration l times before the current iteration. For the two first iterations, α_{mean} was taken as $10 rad/s^2$. In is the vector containing the momentum of Inertia of each joint, which can be estimated with regard to equation 4.14, as τ_{max} have their values specified in the constraints of the scene in CoppeliaSim. Therefore, a inequality is formulated as

$$-\frac{\Delta t}{In}\tau_{max} + \dot{\theta}_{(k-1)} \leq \dot{\theta} \leq \frac{\Delta t}{In}\tau_{max} + \dot{\theta}_{(k-1)} \tag{4.15}$$

which is adapted to the form of matrix equations so the matrices A and b in 4.15 are established. In the case of the MPC method, these matrices are constructed in a more complex way, as it was considered this restraint of mean angular acceleration between two predicted joint velocities, which is an essential idea to make the predictive control work well.

As for the sinusoidal trajectory, this estimation of the momentum of inertia didn't work well, as it is only an approximation and it depends on the joint configuration. A value that was empirically good was the acceleration of $10 rad/s^2$.

5 Results and Evaluation

Different simulations were made to test the two controllers involved in this project. Two kinds of trajectories were approached: a circular and a sinusoidal one, both tested for distinct assigned periods to be fulfilled. These paths can show the effectiveness of the trajectory controllers, considering the nature of their curves.

One section was made, in this chapter, for each trajectory to evaluate the manipulator performance with the two controllers applied, both with time period specified of two seconds. It is shown in figures and discussed the path obtained, the control actions, the angular positions and torques applied to each joint. The number of predictions of the MPC approached in these sections was $p = 10$. The control horizons used in all the situations for the MPC was $\tau_p = 20T = 1s$.

In the last section, the results of mean scale, error and control action are exhibited and discussed for all the different periods assigned to fulfill the task, as also the two MPC controllers with $p = 6$ and $p = 10$.

5.1 Circular Trajectory

The circular trajectory was performed in the XY plane, keeping the Z coordinate fixed. It is convenient to define the trajectory with respect to the origin of the reference coordinate frame. The parametric equation that defines the translation that corresponds to a circumference with radius R and centered at the origin of the reference frame can be expressed as

$$tr = 0 + R\,cos(\,\Omega\,\sigma(t)\,)\hat{\imath} + R\,sin(\,\Omega\,\sigma(t)\,)\hat{\jmath} + 0\hat{k} \tag{5.1}$$

wherein tr is a quaternion, $\sigma(t)$ is the function formerly defined as the dilated time function and Ω is the angular frequency. Considering a reference trajectory that has the same orientation as the reference frame, the rotation quaternion is, hence, given by

$$rot = 1 + 0\hat{\imath} + 0\hat{\jmath} + 0\hat{k} \tag{5.2}$$

so the dual quaternion $\underline{traj} = rot + 0.5\epsilon(\,tr\,rot\,)$ can determine the requested path.

To express the trajectory dual quaternion pose $\underline{x}_d$ in relation to the robot's base, it is necessary to perform the multiplication

$$\underline{x}_d = \underline{b}_p^* \underline{ref}_p \underline{traj}(\sigma)$$

$$\underline{x}_d = \underline{b}_p^* \underline{ref}_p \,(\,1 + \epsilon(\,R/2\,cos(\Omega\sigma)\hat{\imath} + R/2\,sin(\Omega\sigma)\hat{\jmath}\,)\,)$$

then, finally, $vec_8(\underline{x}_d)$ can be the reference signal of the desired pose provided to the controller. The pose velocity is determined as

$$vec_8(\underline{vx}_d)(\sigma) = \frac{\mathrm{d}(\, vec_8(x_d)\,)}{\mathrm{d}\sigma}$$

$$vec_8(\underline{vx}_d)(\sigma) = vec_8(\, \underline{b}^*_p \, \underline{\underline{ref}}_p \, \epsilon(\, -\Omega R/2\, sin(\Omega\sigma)\hat{\imath} + \Omega R/2\, cos(\Omega\sigma)\hat{\jmath}\,)\,)$$

For the local and MPC methods, a tuning rule that worked well for the proportional gain to perform the circle trajectory was $K = 200 \cdot 2\pi/\Omega$, as $2\pi/\Omega$ is the period of the assigned task. The circle trajectory tested that is considered in this section is the one with radius $R = 0.1m$ and nominal period of 2 seconds.

5.1.1 Data Analysis

In figures 9 and 10, it is exhibited the measured points obtained (in meters), respectively, by the MPC and Local methods. In blue, it is shown the expected trajectory for the end-effector, while the symbol in red shows the measured points that were achieved. It is evident that the MPC method worked much better than the Local method at preserving the original circular path. The trajectory started at the point on the far right (as in a regular trigonometric cycle). Until the third quarter of the circumference, the Local method seemed to be working well, even better than the MPC in terms of scaling (it is possible to observe that the points are less spaced). In the third quarter, however, the Local method deformed the path considerably (which also harmed the scaling) as the MPC was able to manage to keep a low error. It is also possible to verify that the Local method produced an increasing error at the end of the last quarter.

The figures 11 and 12 show the obtained trajectories for, respectively, the MPC and the Local methods. The pictures help to reiterate that the Local method noticeably deformed the original path while the MPC was efficient to preserve it. The plots in three dimensions also showed that, in the third quarter or the circular trajectory, the Local method produced a substantial error along the direction of the Z axis, and a smaller yet perceptible error in this vertical direction at the end of the path.

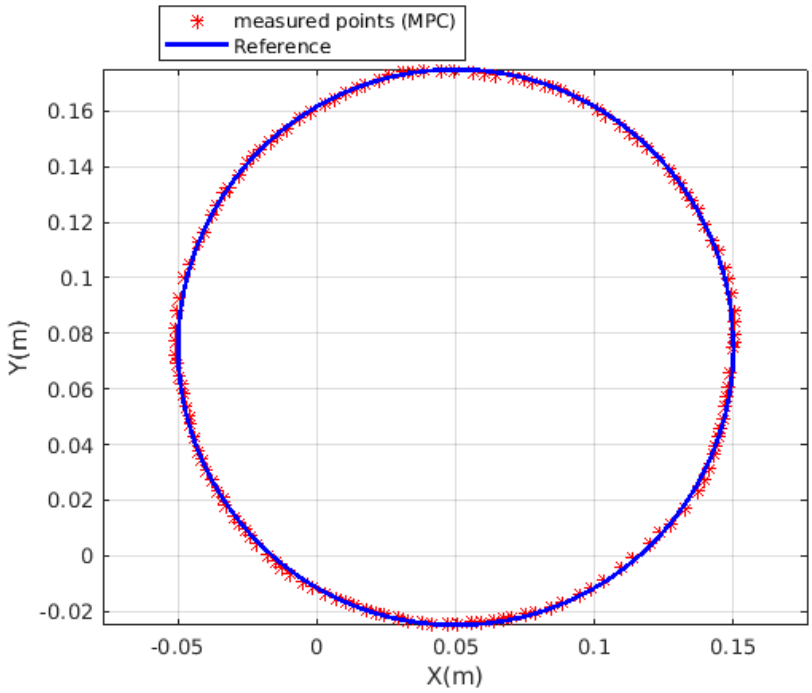

Figure 9 – 2D plot of the circle Trajectory obtained for the MPC method, with period of 2 seconds.

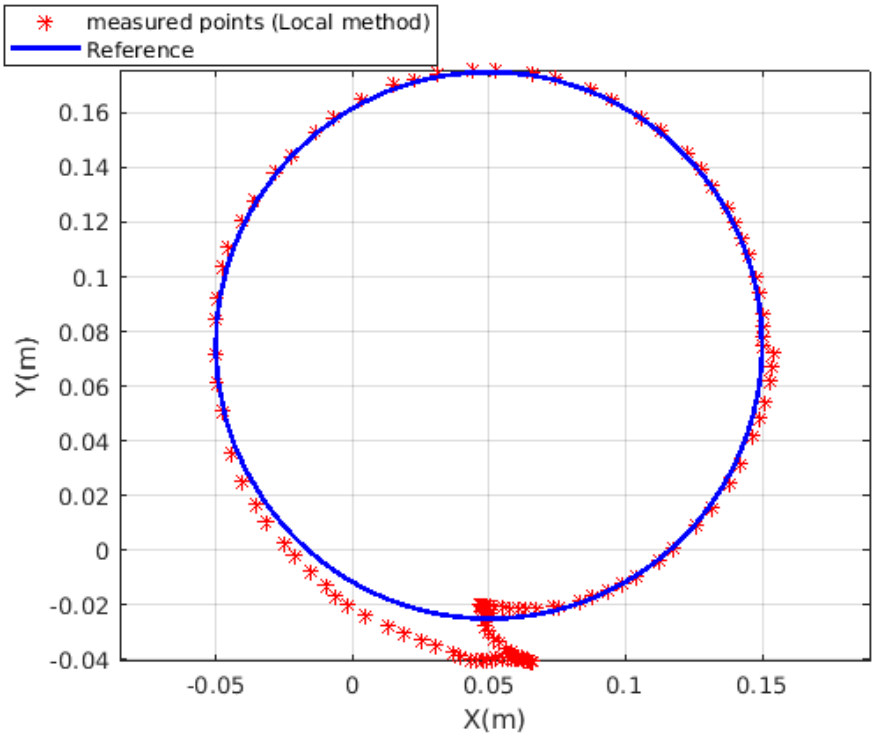

Figure 10 – 2D plot of the circle Trajectory obtained for the Local method, with period of 2 seconds.

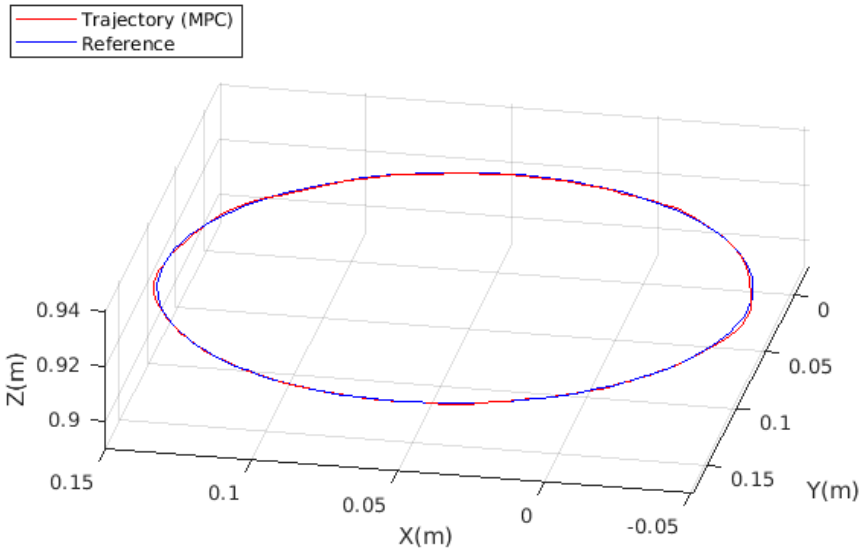

Figure 11 – 3D plot of the circle Trajectory obtained for the MPC method, with period of 2 seconds.

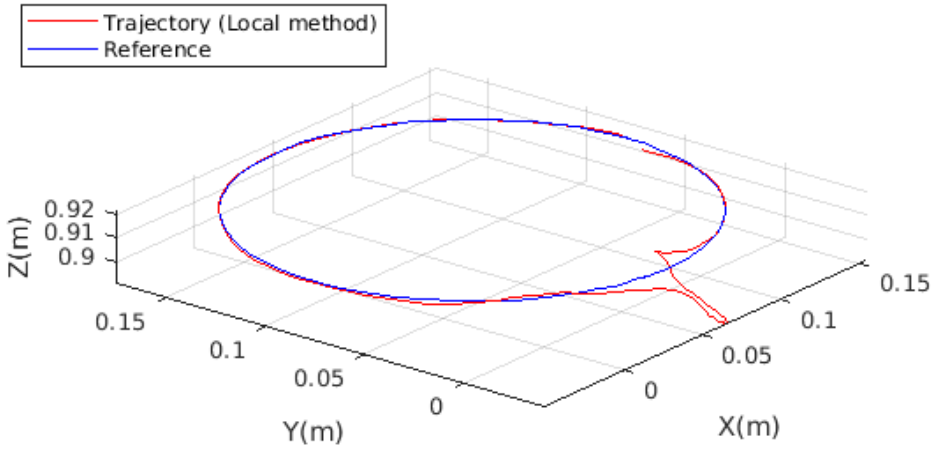

Figure 12 – 3D plot of the circle Trajectory obtained for the Local method, with period of 2 seconds.

It is also interesting to analyze the behavior of each element of the pose (the three degrees of freedom of translation and the three ones of rotation). Figures 13 and 14 show the translation coordinates (in meters) in function of $\sigma(t)$ (in seconds). It is visible to confirm the

previous analysis about the effectiveness of MPC, oppositely to the Local method, in regard of the third quarter of the cycle (around $\sigma = 1.5s$) and the end of the fourth quarter (around $\sigma = 2s$) , wherein it is possible to verify the discrepancy between the reference function (in blue) and the measured coordinates (in red). Both the MPC and Local methods presented problem to keep a low error for Y in the beginning of the trajectory, which highlights the difficulty of using the kinematic control with constraints that may not be convenient for certain states. It may seem that the Z coordinate in the graphs has undergone a larger variation, but its scale is very different from the ones taken into account in the two graphs for the coordinates X and Y. Nevertheless, the noisy function obtained shown in figure 13 highlights what may be the small inaccuracies from the QP solver used or limitations of the joint control.

In regard to rotation degrees of freedom, graphs of spin coordinates are shown in figures 15 and 16, in reference to the MPC and Local methods, respectively. It is important to evaluate its behavior, since in the trajectories plots, it is not possible to observe the orientation behavior. It can be verified that even the orientation coordinates obtained (in red) diverged greatly from the reference (in blue) for the local method around $\sigma = 1.5s$ (same period of fail for the translation coordinates), while the error for the MPC method remained in a smaller range along the whole period. Despite that, the rotation coordinates where similar. The noise may also be a consequence from the restraints of joint control and numerical inaccuracies from the QP solver used.

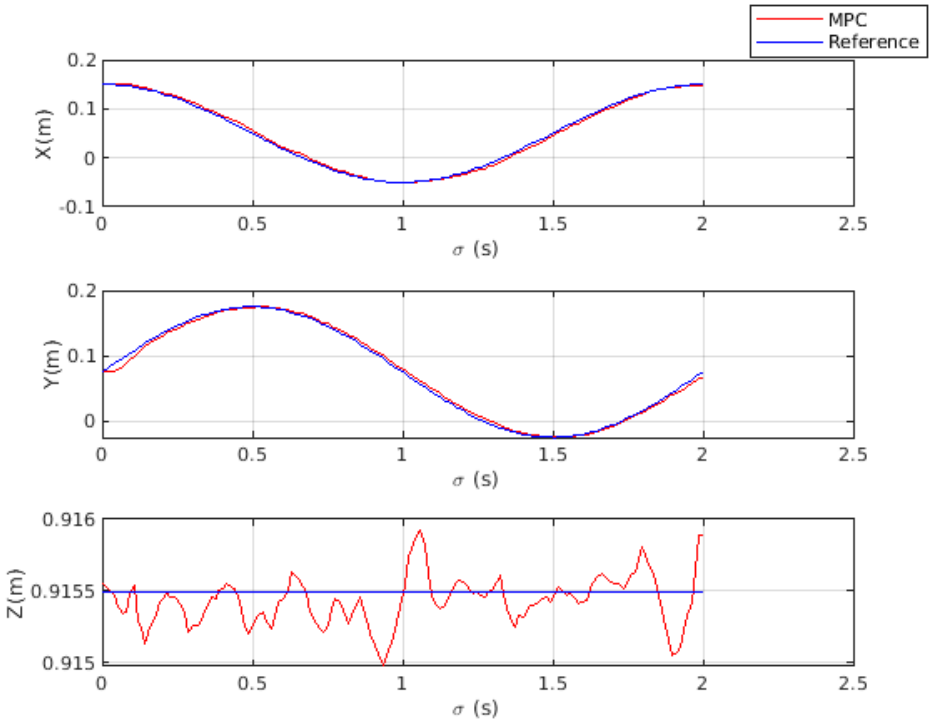

Figure 13 – Behavior of each translation element.

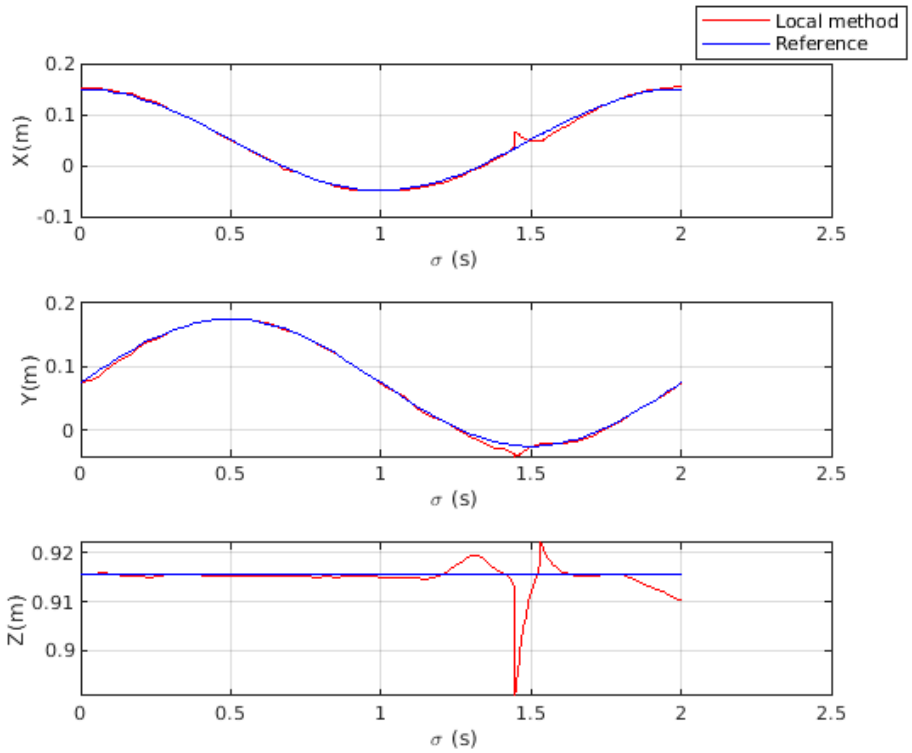

Figure 14 – Behavior of each translation element for the Local Method.

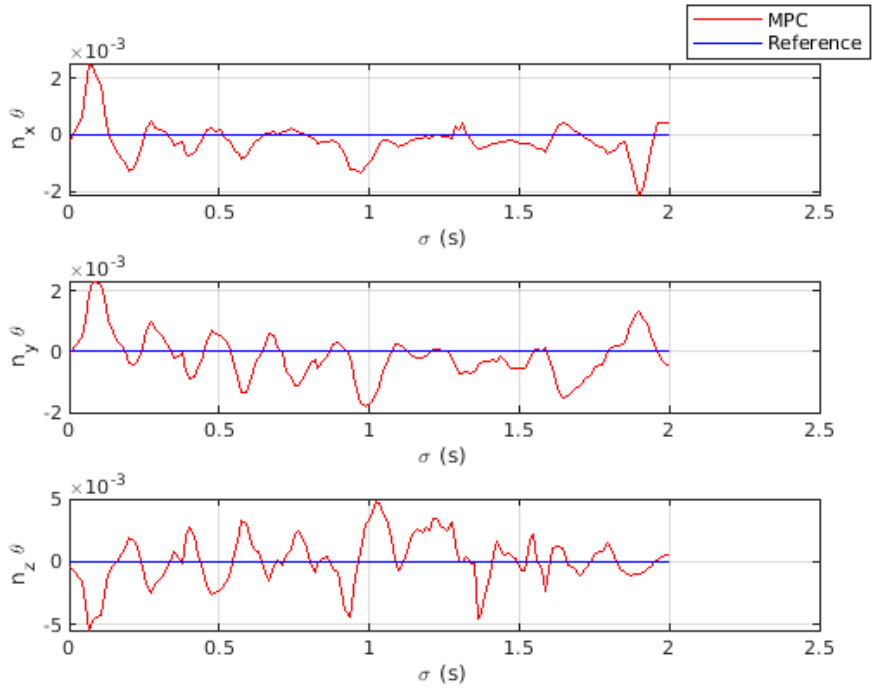

Figure 15 – Behavior of each rotation element for the MPC method.

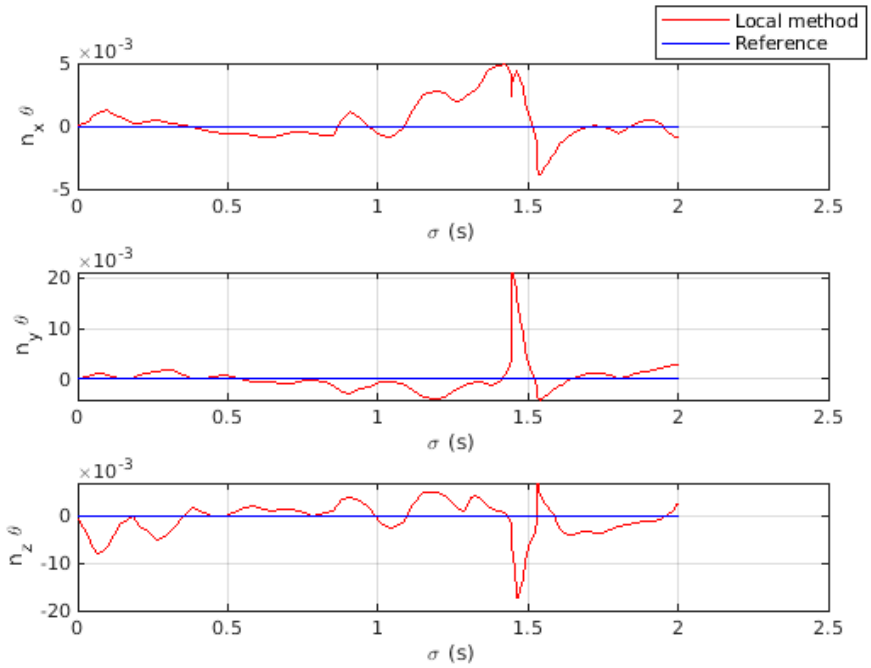

Figure 16 – Behavior of each rotation element for the Local method.

Although the same QP solver is used in the two methods compared, they produced very different control signals in order to perform the same trajectory in the nominal time of 2 seconds. Figures 17 and 19 show the joint velocities for the MPC method and figures 18 and 20 show the joint velocities for the Local method, all in function of $\sigma(t)$ (in seconds). The joint velocities are given in rad/s. It is expected the joint velocities measured (in red) to be delayed by one sampling time $T = 0.05s$ from the reference joint velocities (in blue), which, in general, occurred. However, it is noticeable that, in some periods, the measured joint velocities did not follow their reference, which clearly exposes the limitations imposed by the joint control in the cascade. It is interesting to notice that, right around $\sigma = 1.5s$ the joint velocities in the local (specially the joints 1 to 4) were submitted to large variations, in order to recover from the significant error generated at this period. It can also be verified that, from one sampling time to another, in general, the joint velocities varied more for the MPC method, as in every instant, this procedure makes a different prediction that depends on the approximation of future Jacobians, so the strategy tend to be more susceptible to changes.

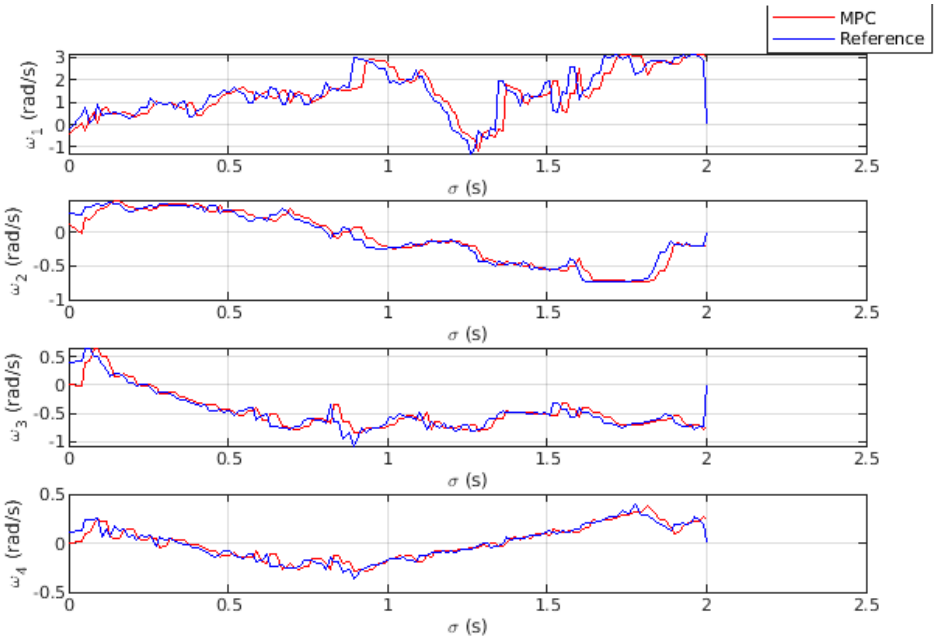

Figure 17 – Behavior of each control action for the MPC method (from joint 1 to 4).

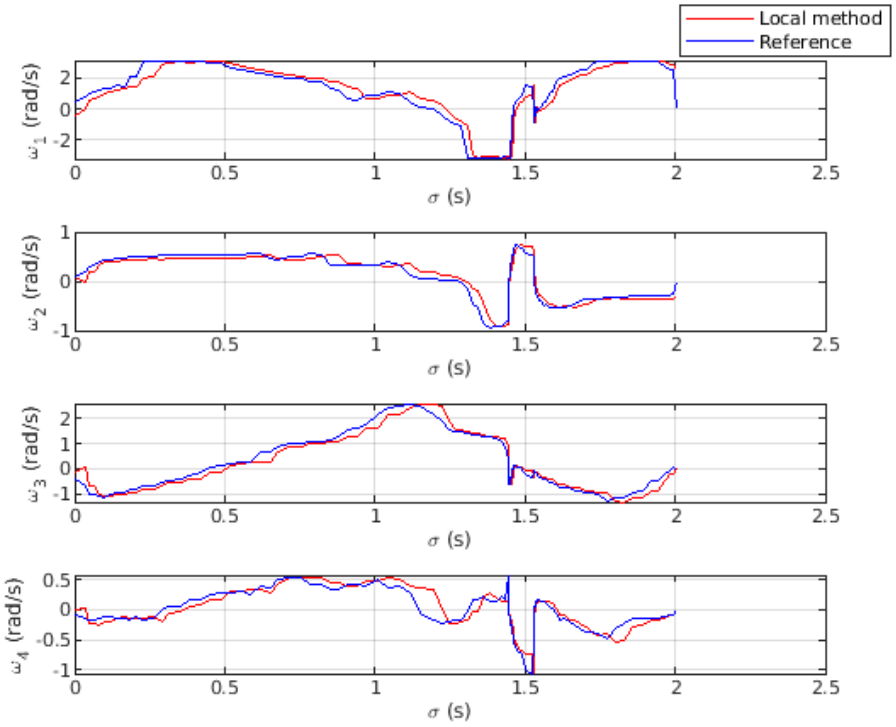

Figure 18 – Behavior of each control action for the Local method (from joint 1 to 4).

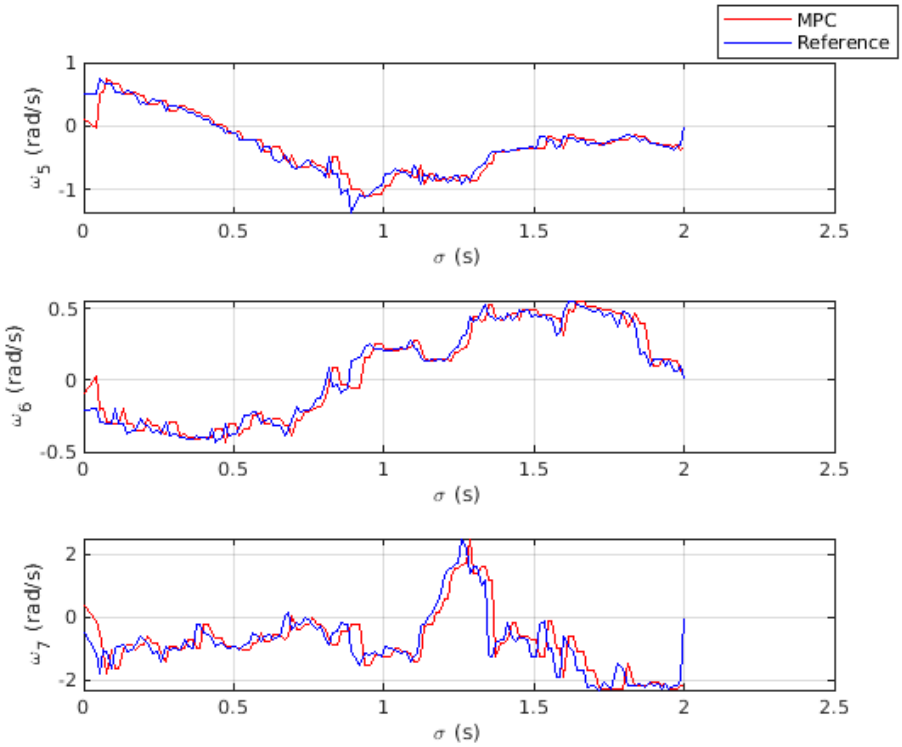

Figure 19 – Behavior of each control action for the MPC method(from joint 5 to 7).

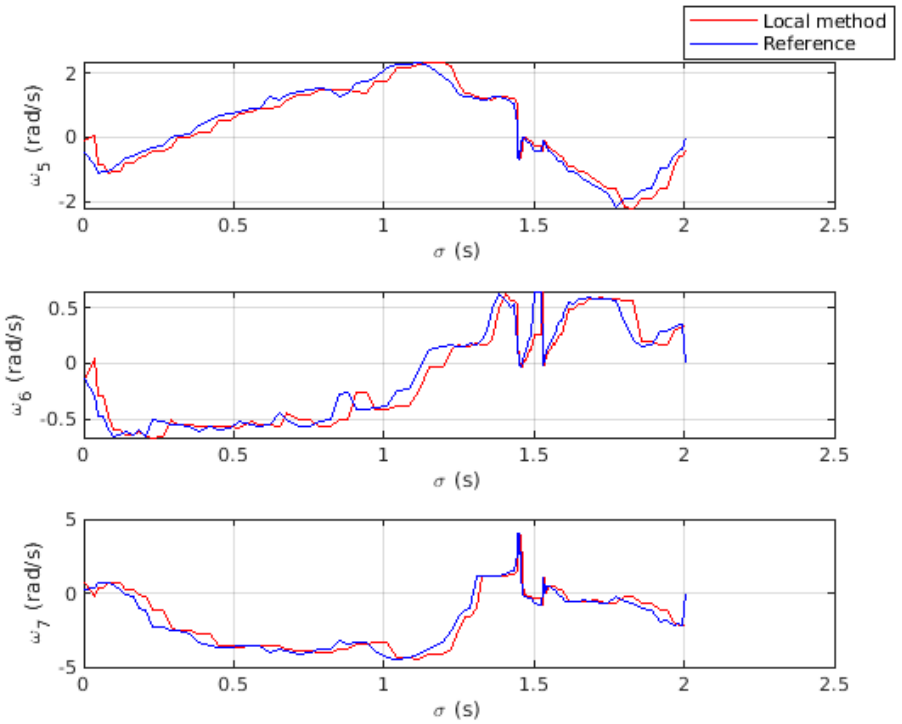

Figure 20 – Behavior of each control action for the Local method (from joint 5 to 7).

As the joint velocities have different outlines from one method to the other, consequently the angular positions obtained will also differ, which is exhibited in figures 21

and 22. In green, it is illustrated the joint angular positions obtained through the MPC and in magenta, the ones obtained through the Local method.

To obtain these different profiles of velocity and angular position, certainly the two methods needed to apply distinct torques, which is shown in figures 23 and 24. In general, the torques of MPC (in green) and the Local method (in magenta) remained in similar ranges. However, for the joints 4 and 5, there were peaks of torque from the Local method that were very far from the maximum magnitude of the MPC torque peaks.

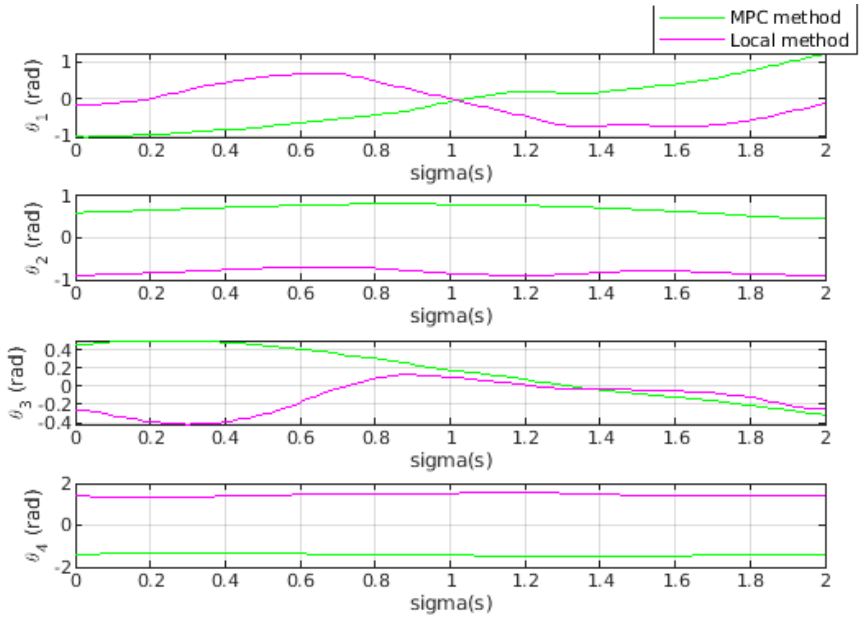

Figure 21 – Behavior of each joint angle (from joint 1 to 4).

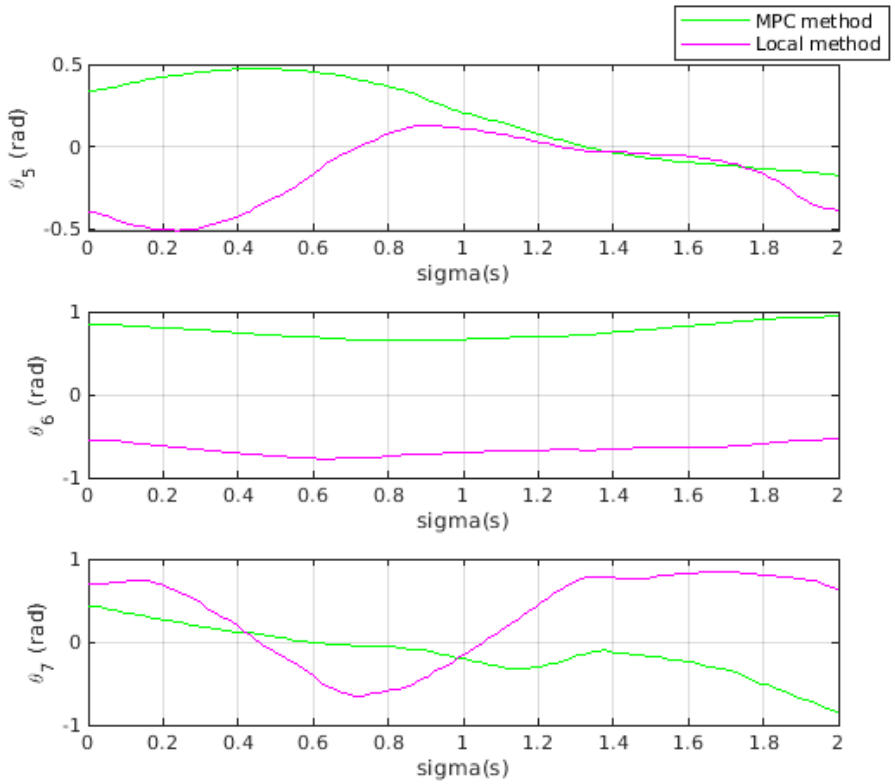

Figure 22 – Behavior of each joint angle (from joint 5 to 7).

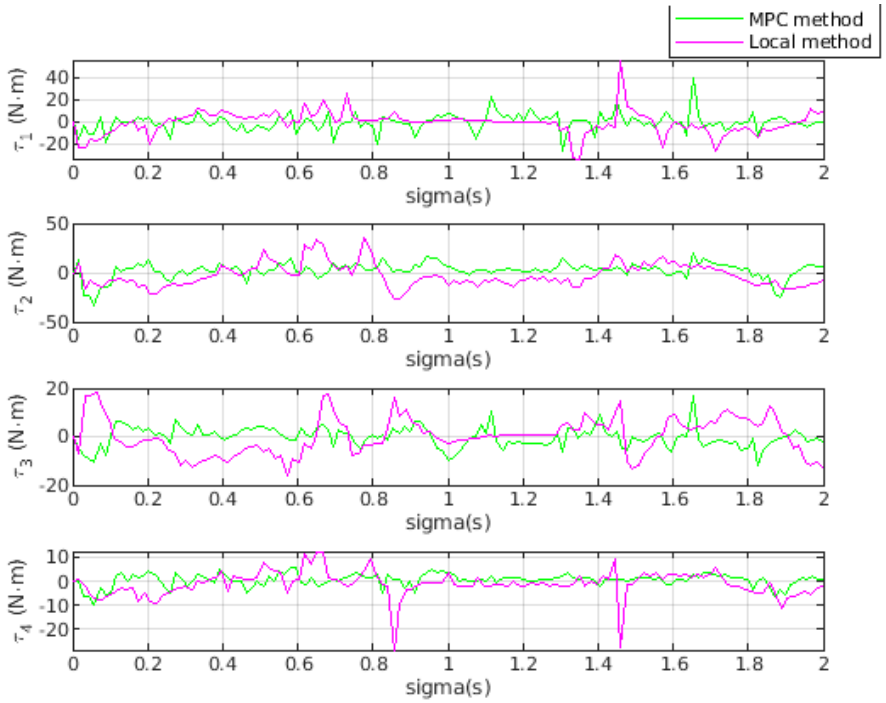

Figure 23 – Behavior of each applied torque (from joint 1 to 4).

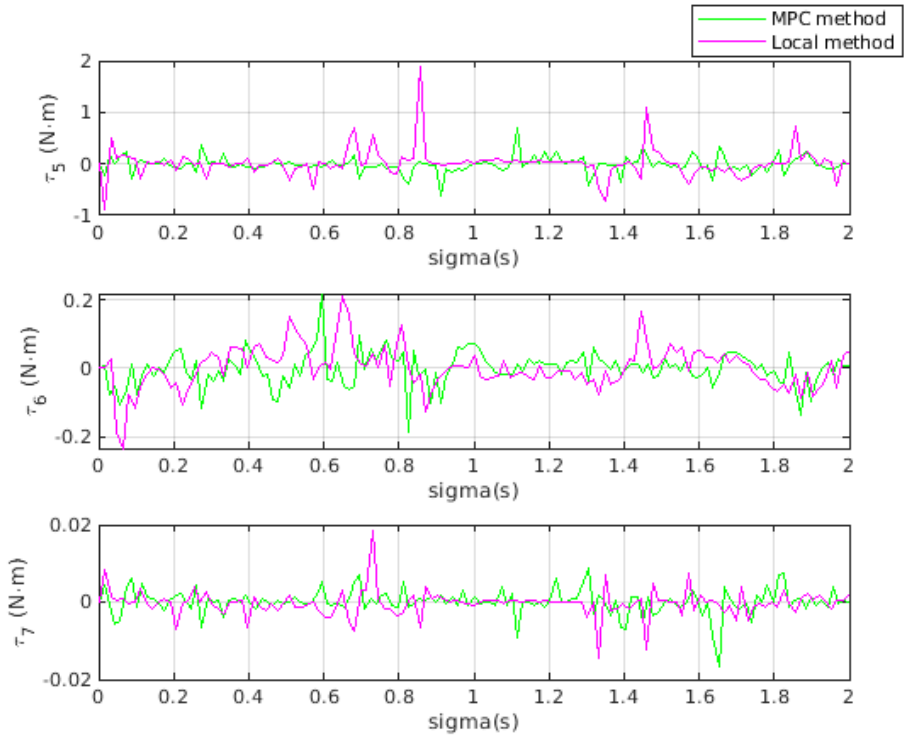

Figure 24 – Behavior of each applied torque (from joint 5 to 7).

5.2 Sinusoidal Trajectory

The sinusoidal trajectory was performed in the XZ plane (keeping the Y coordinate fixed), also defined in relation to the reference frame. The parametric equation that defines the translation corresponding to the sinusoidal curve of Z in function of X with amplitude Amp can be defined as

$$tr = 0 + v\,\sigma(t)\hat{\imath} + 0\hat{\jmath} + Amp \cdot sin(\,\Omega v\sigma(t)\,)\hat{k} \tag{5.3}$$

wherein tr is a quaternion, $\sigma(t)$ is the dilated time function, Ω is an angular frequency in terms of the X coordinate and v is the linear velocity of X (associated to $\sigma(t)$).

The reference of this trajectory also has the same orientation as the reference frame. Therefore, rotation quaternion is expressed as in equation 5.2, with the dual quaternion $\underline{traj} = rot + 0.5\epsilon(\,tr\,rot\,)$ representing the desired path. Hence, the dual quaternion $\underline{x}_d$ associated to the required pose, in relation the robot's base, is determined as

$$\underline{x}_d = \underline{b}_p^* \underline{ref}_p \left(1 + \epsilon\left(\frac{v\sigma}{2}\hat{\imath} + \frac{Amp}{2}\,sin(\Omega v\sigma)\hat{k}\right)\right) \tag{5.4}$$

then, $vec_8(\underline{x}_d)$ is the reference vector of the controller. The pose velocity $\underline{vx}_d$ is, therefore,

given by

$$vec_8(\underline{vx}_d)(\sigma) = \frac{\mathrm{d}(\, vec_8(x_d)\,)}{\mathrm{d}\sigma}$$

$$vec_8(\underline{vx}_d)(\sigma) = vec_8\left(\underline{b}_p^* \underline{ref}_p \left(1 + \epsilon\left(\frac{v}{2}\hat{\imath} + \frac{Amp\,\Omega}{2} cos(\Omega v\sigma)\hat{k}\right)\right)\right)$$

For the Local and MPC methods, a tuning rule that worked well for the proportional gain to perform the sinusoidal trajectory was $K = 500/P$, wherein P is the period of the task, imposed to be $2\pi/(15v)$. The sinusoidal trajectory tested that is approached in this section is the one with $Amp = 0.1m$ and a nominal period of 2 seconds. Hence, the gain used was $K = 250$. The values of v and Ω were chosen to be $\pi/15$ and 60, respectively, which produce a sinusoidal path with curves that have great inflection, which is a good way to test the controllers and the quality of the prediction of MPC.

5.2.1 Data Analysis

The measured points in X and Z coordinates obtained by the MPC and Local methods are shown, respectively, in figures 25 and 26. In blue, it is shown the expected trajectory for the end-effector, while the asterisk in red shows the measured points that were reached. Also for the sinusoidal trajectory, the MPC method presented much better results than the Local method, as it is clear the predictive one was able to preserve the original path, oppositely to the the purely local. Right close the curves in the peaks, it is evident that the Local method presented its limitations, which were overcame by the MPC, that was able to predict well the accomplishment of those sharp turns. The points around the two last valleys were the ones that presented the largest errors of the Local method. It can be verified that, for the MPC (figure 25), the points are spaced before reaching the curve in the peak, oppositely to when the curve in being made, which denotes that the scaling factor decreased (for a good reason) right before these regions.

In figures 27 and 28, it is shown, respectively, the 3d plot of the trajectory obtained through the MPC and Local methods. These pictures shows that, in addition to errors in X and Z, the peak curves were a source of significant errors in Y for the Local method, while the MPC method worked well to keep a low error through the whole path.

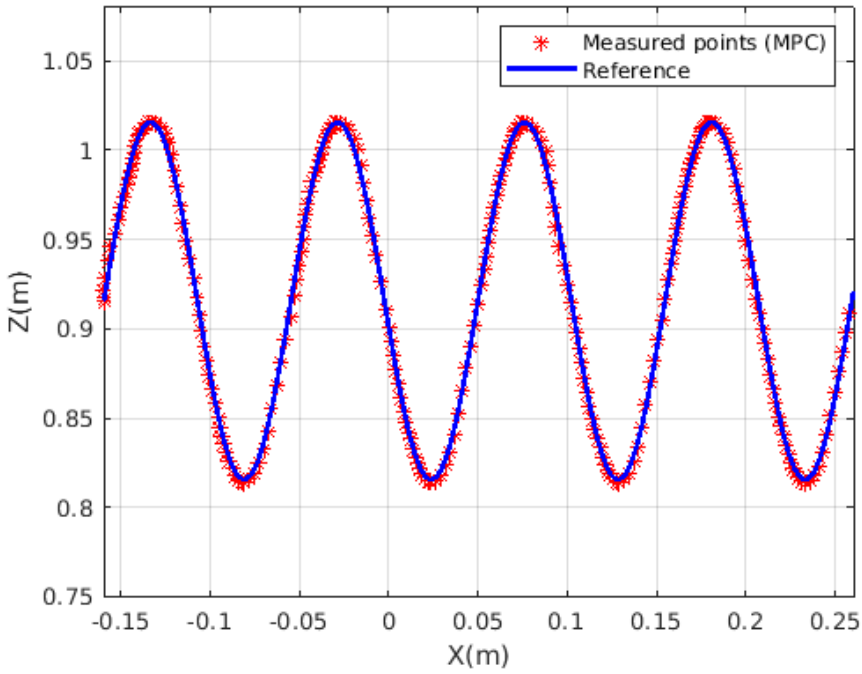

Figure 25 – 2D plot of the circle Trajectory obtained for the MPC method, with period of 2 seconds.

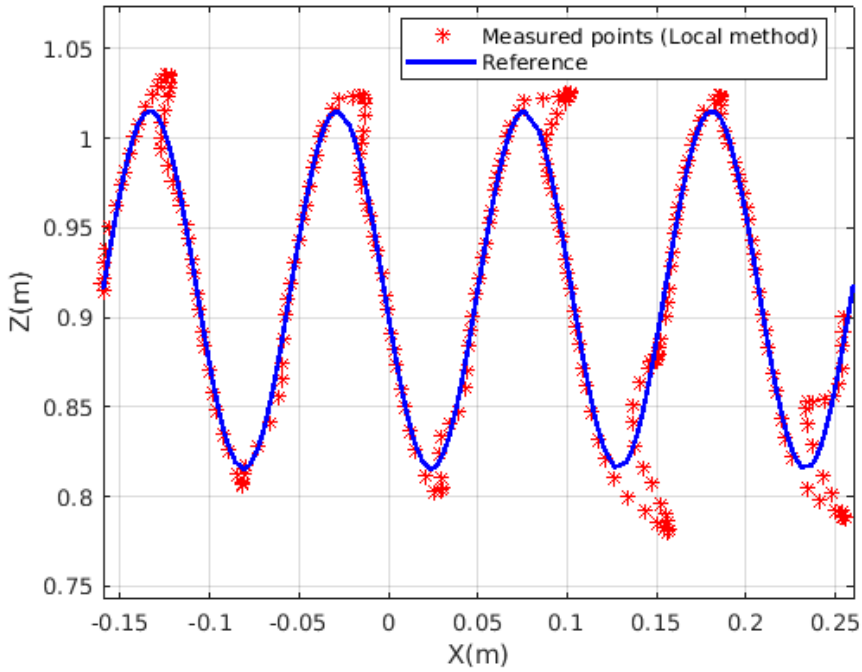

Figure 26 – 2D plot of the circle Trajectory obtained for the Local method, with period of 2 seconds.

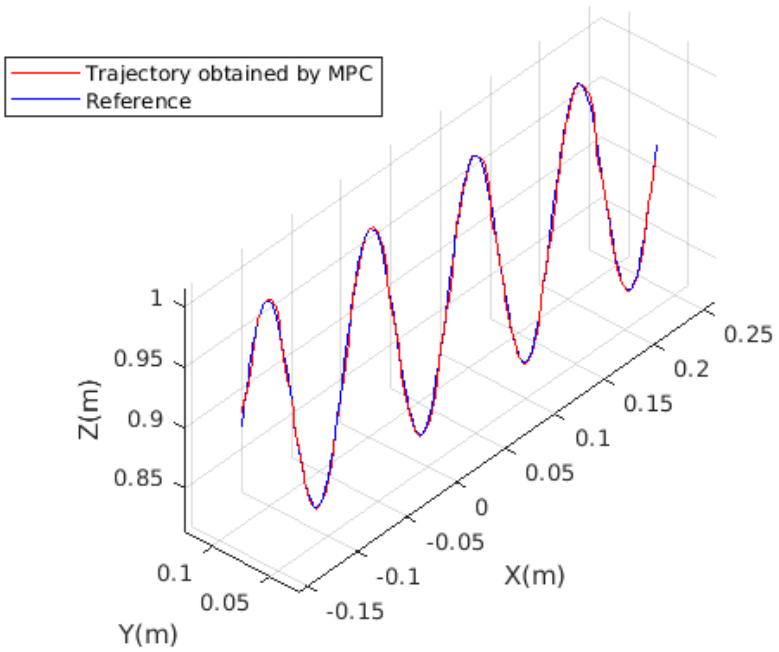

Figure 27 – 3D plot of the circle Trajectory obtained for the MPC method, with period of 2 seconds.

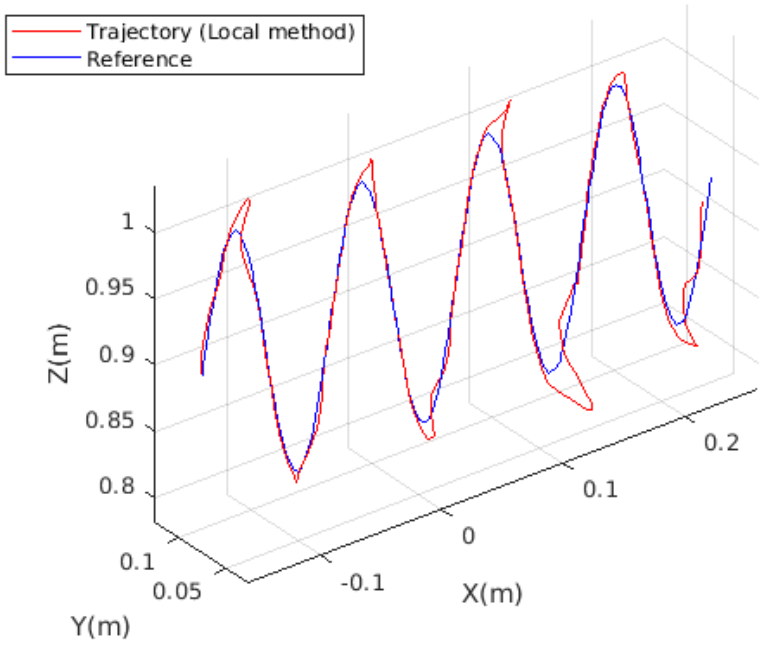

Figure 28 – 3D plot of the circle Trajectory obtained for the Local method, with period of 2 seconds.

The six elements of the pose were also taken into account in the sense of showing their evolution through the dilated time function. Figures 29 and 30 illustrate the translation elements (in meters) in function $\sigma(t)$ (in seconds). In red, the measured translation coordinates are shown and, in blue, their references. It is reiterated, observing the pictures, that

the Local method presented larger errors, specially near to the curves in the peaks of the sine reference. It might seem that the error of Y for the MPC was larger, but it is necessary to pay attention to the scale in the axis of the plot. Nevertheless, this noisy curve obtained (even with a small variance of a maximum of 2mm) highlights the limitations already mentioned of the joint control and/or the QP solver.

In figures 31 and 32, the rotation degrees of freedom obtained through the MPC and Local methods, respectively, are shown. Right before $\sigma = 1.5s$, the obtained rotation coordinates (in red) for the Local method presented a high peak, greatly above their averages. The MPC method produced a small variation (observing the scale of 10^{-3}), as a consequence of the limitations of the resolution.

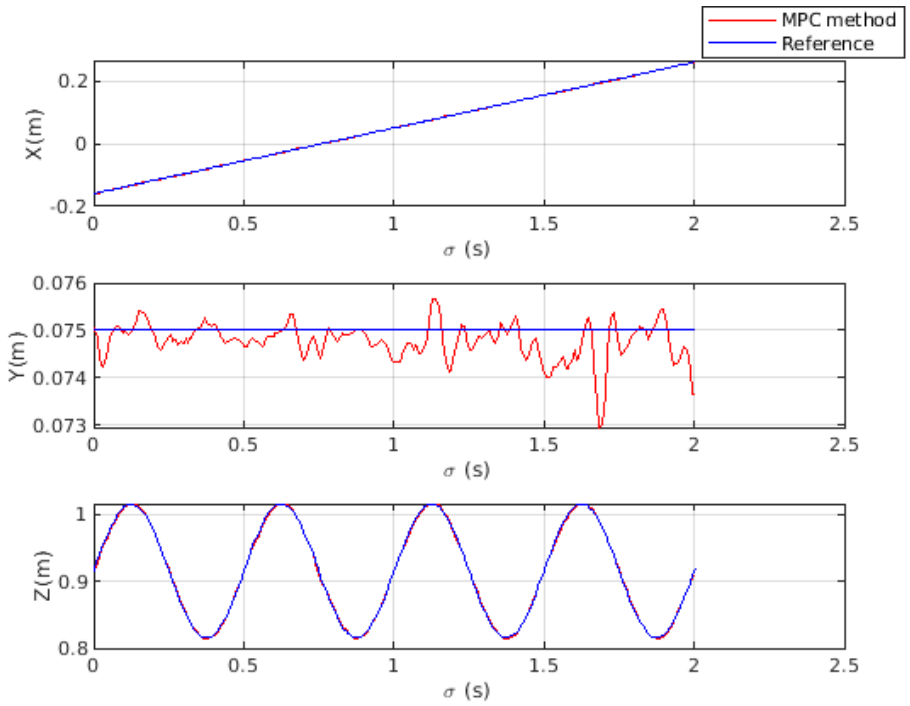

Figure 29 – Behavior of each translation element.

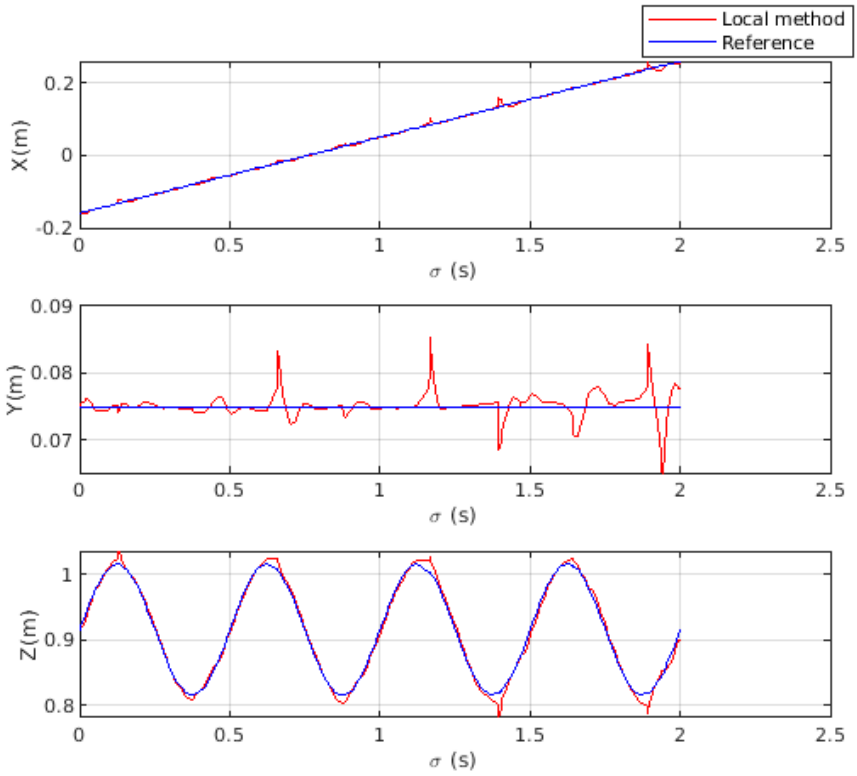

Figure 30 – Behavior of each translation element for the Local Method.

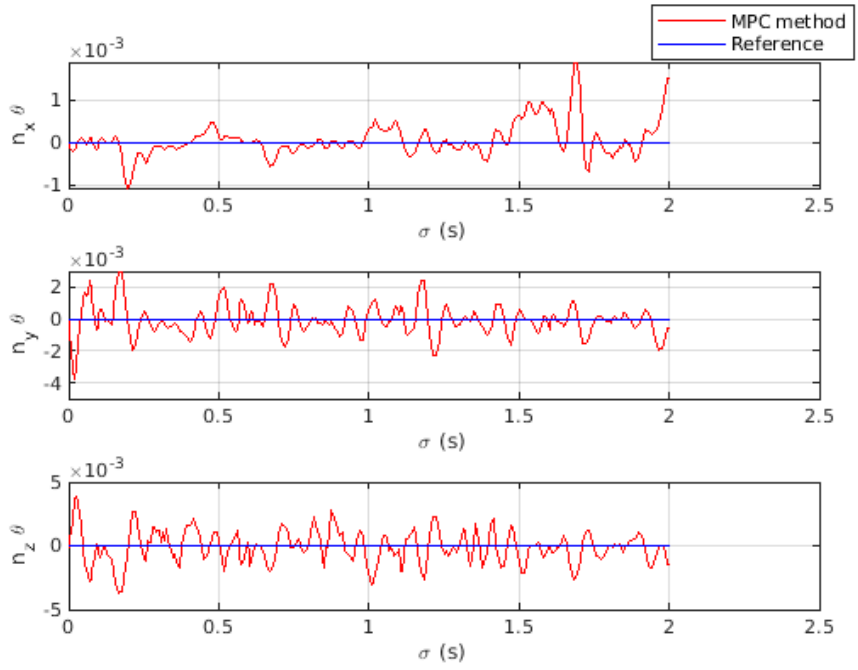

Figure 31 – Behavior of each rotation element.

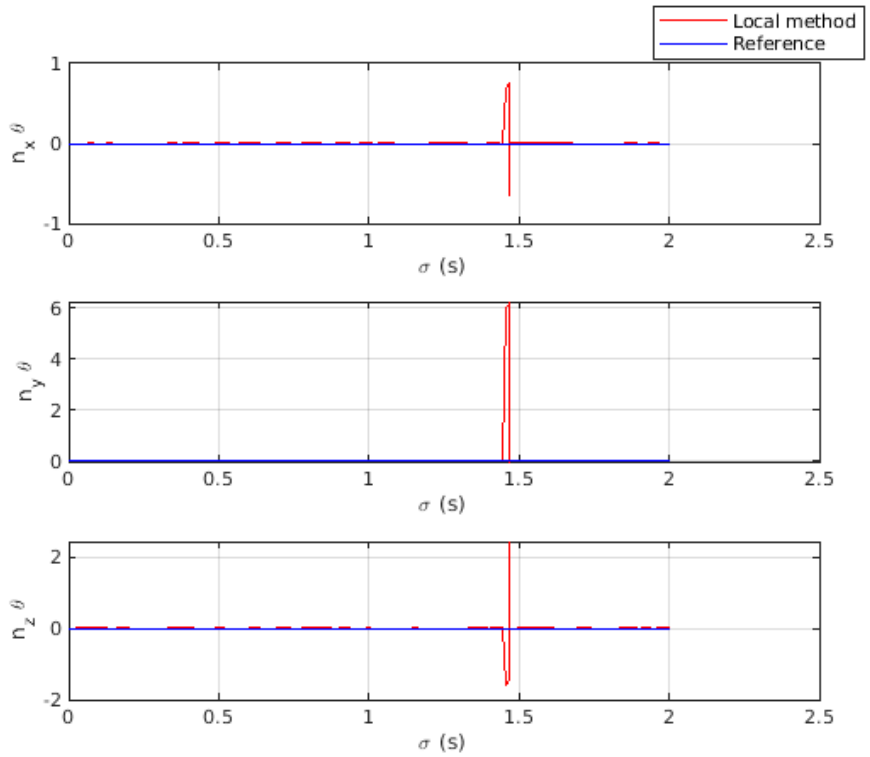

Figure 32 – Behavior of each rotation element for the Local method.

For the sinusoidal trajectory, it is also possible to observe that the two methods taken into account demanded considerably distinct control actions to fulfill the same task, although there are some peculiar similarities. Figures 33 and 35 show the joint velocities(in rad/s) for the MPC, which are shown, for the Local method, in Figures 34 and 36, all in function of $\sigma(t)$ (in seconds). In general, the measured joint velocities were delayed in relation to the reference by one sampling time of $T = 0.05s$. In some points, however, it is verifiable that the reference was not followed with null error, again exposing the limitations of joint control. One peculiar similarity between the methods is that the joint velocities 2, 4 and 6 presented a certain periodicity for the two methods, even though the functions obtained for each joints is different from one method to another. Also for this trajectory, the MPC method produced a larger variation in the control signal from one sampling time to another.

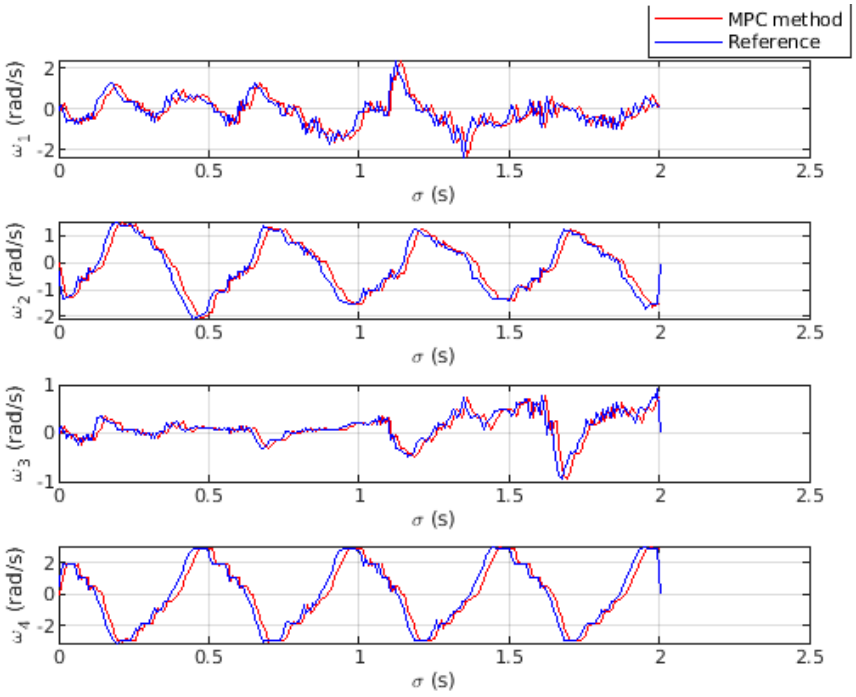

Figure 33 – Behavior of each control action (from joint 1 to 4).

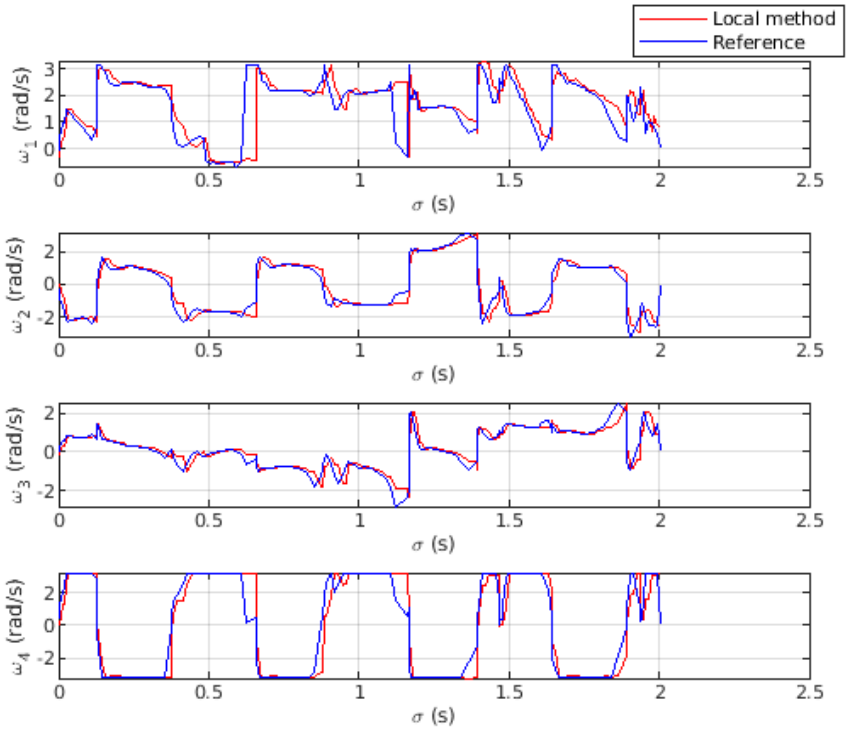

Figure 34 – Behavior of each control action for the Local method (from joint 1 to 4).

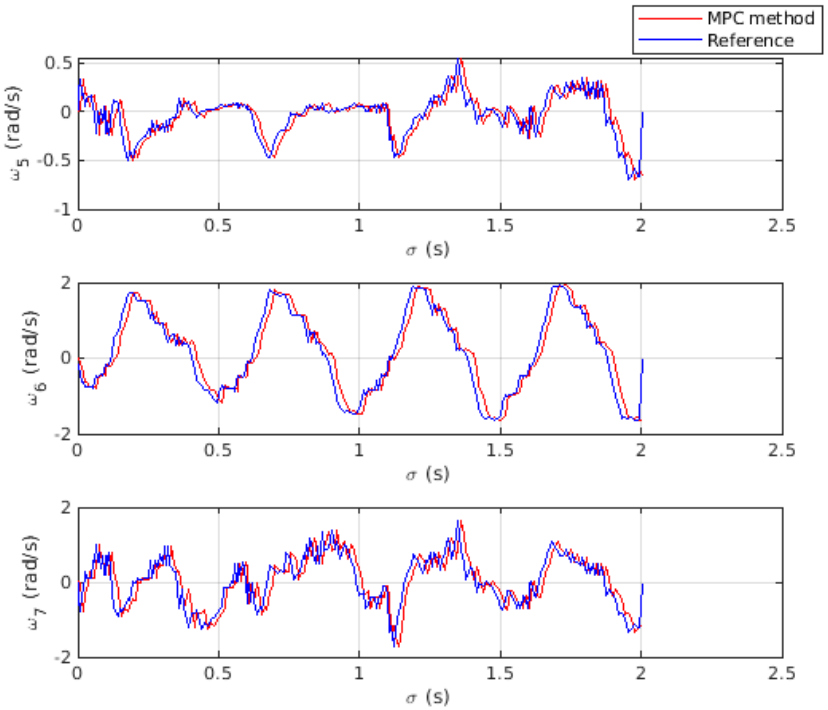

Figure 35 – Behavior of each control action (from joint 1 to 4).

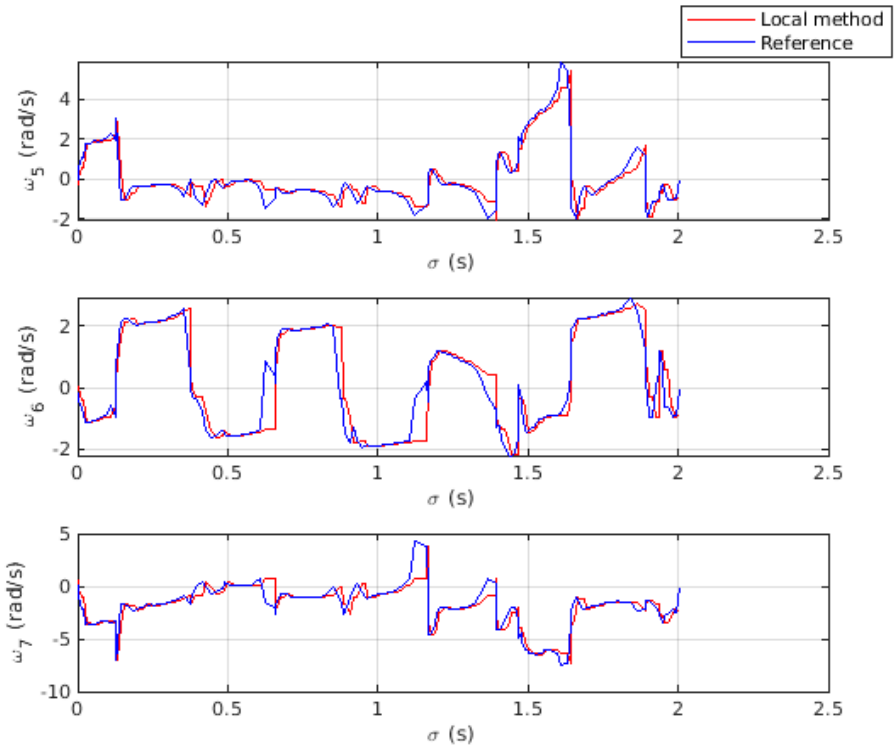

Figure 36 – Behavior of each control action for the Local method (from joint 1 to 4).

Figures 37 and 38 show the joint angular positions measured(in rad) for the MPC (in green) and Local (in magenta) methods. As expected, the angular positions differ from

one method to the other, although the similarity of the nearly periodical behaviour remains for the joints 2, 4 and 6. It is possible to observe that the Local method produced a very abrupt variation in the first and seventh angular positions, near 1.4 seconds, while the MPC obtained ones remained in a much smaller range.

The different applied torques are exhibited in figures 39 and 40. It is very evident that the torques applied for the Local method (in green) presented a larger range of values than the ones obtained through the MPC procedure, which is another advantage of this algorithm, as the trajectory was better fulfilled with less effort.

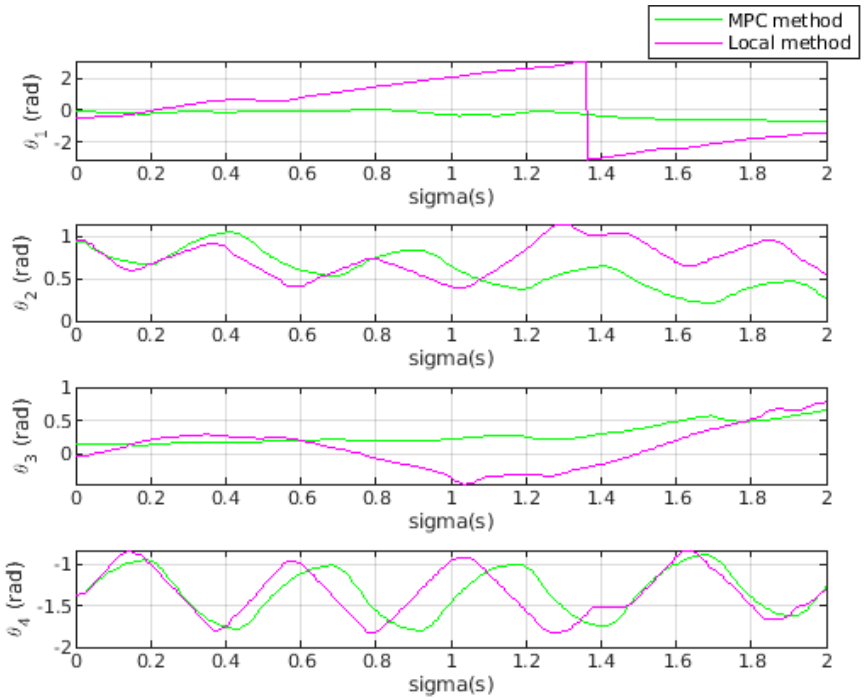

Figure 37 – Behavior of each joint angle (from joint 1 to 4).

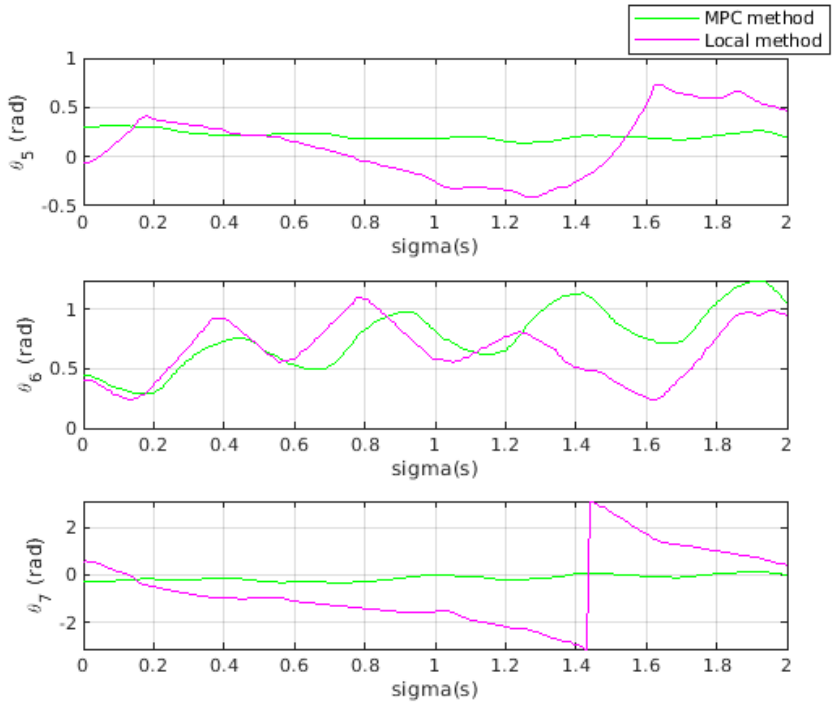

Figure 38 – Behavior of each joint angle (from joint 5 to 7).

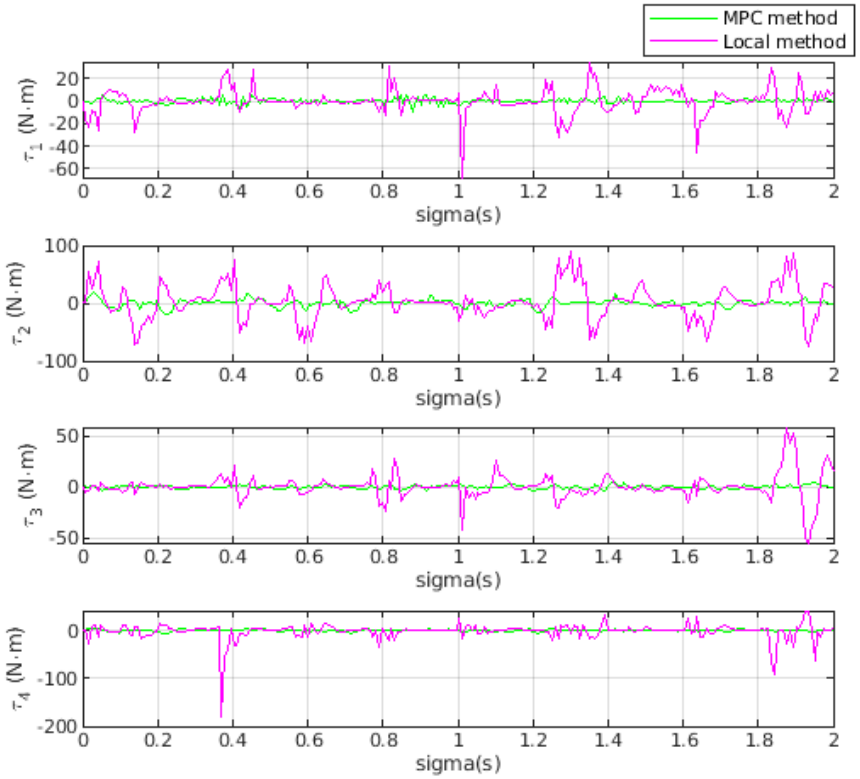

Figure 39 – Behavior of each applied torque (from joint 1 to 4).

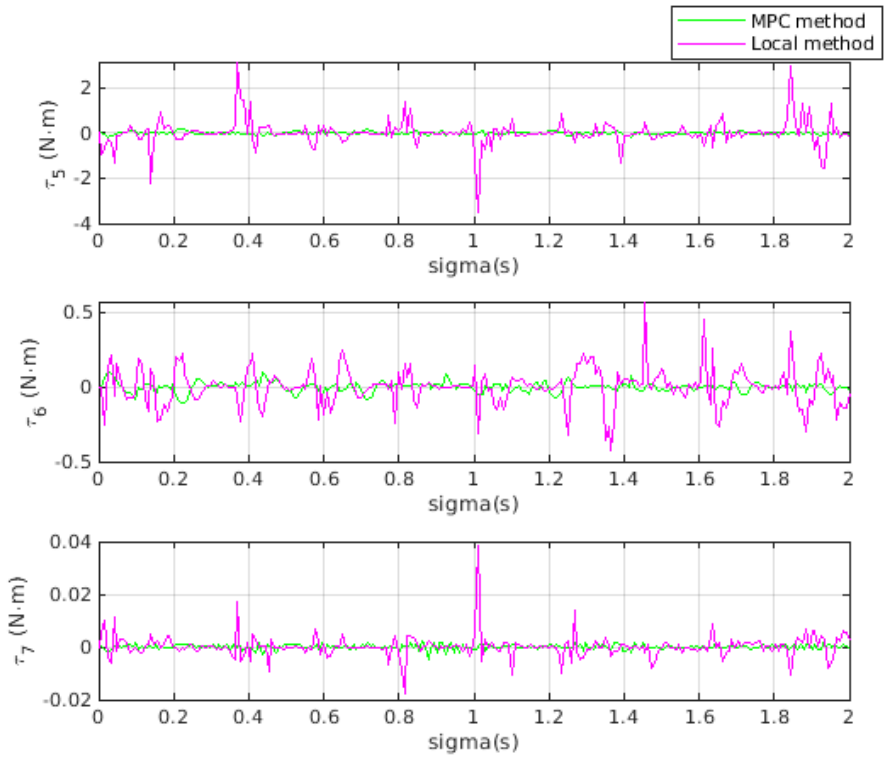

Figure 40 – Behavior of each applied torque (from joint 5 to 7).

5.3 Main Results

Other tests with the controllers were made, in order to test their effectiveness under different conditions. In regard to all the distinct total periods T_{tot} assigned to the circular and sinusoidal trajectories, the table 1 presents the mean error, the mean value of the control actions and the mean value of the scaling factor for the Local method and the MPC methods with $p = 6$ and $p = 10$.

For the circular trajectory, in general, the MPC controller was able to keep a lower error than the Local method. However, for $p = 6$ and the total period of $T_{tot} = 8s$, the MPC produced a larger mean error, while even the mean value of the control action was higher than the obtained through the Local method. The smallest mean values of error and control actions were obtained by the MPC method with $p = 10$, as the higher mean values of the scaling, which shows the superiority of this controller for this application.

As for the Sinusoidal trajectory, lower mean values were also obtained for the error using the MPC controller. Nevertheless, for $T_{tot} = 16s$ and $T_{tot} = 8s$, the MPC controller with $p = 6$ produced higher mean error values than the Local method. Although the scaling factor presented higher mean values for the Local method, it is debatable to assert something about its quality, as there is a trade-off between obtained error and scaling factor. Anyway, it was shown that the Local method produced a large malformation of the path in peaks of the

curves, which is certainly undesirable. In general, the MPC method was able to obtain lower mean values of the control signal.

Circular Trajectory									
	e_{mean}			u_{mean}(rad/s)			s_{mean}		
T_{tot}(s)	Local	$p=6$	$p=10$	Local	$p=6$	$p=10$	Local	$p=6$	$p=10$
8	$3 \cdot 10^{-3}$	$3.4 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	1.5427	1.7393	1.8393	0.9785	0.9906	0.9927
4	$3.36 \cdot 10^{-2}$	$3.2 \cdot 10^{-3}$	$2.5 \cdot 10^{-3}$	3.6286	2.8961	2.6303	0.4786	0.5639	0.5714
2	$1.44 \cdot 10^{-2}$	$3.2 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$	3.4070	2.6016	2.8281	0.1789	0.2984	0.3640
1	$8.23 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	$2.6 \cdot 10^{-3}$	4.0542	3.0039	3.0258	0.0418	0.1990	0.2285
Sinusoidal Trajectory									
	e_{mean}			u_{mean}(rad/s)			s_{mean}		
T_{tot}(s)	Local	$p=6$	$p=10$	Local	$p=6$	$p=10$	Local	$p=6$	$p=10$
16	$2.7 \cdot 10^{-3}$	$3.1 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	2.1787	2.3175	1.7413	0.9996	0.9400	0.9863
8	$2.6 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	4.0662	3.1583	2.6165	0.7573	0.5938	0.6598
4	$4.4 \cdot 10^{-3}$	$3.3 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	4.3542	2.5370	2.4295	0.3394	0.2444	0.3196
2	$8.57 \cdot 10^{-2}$	$3.0 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	5.0194	2.4326	2.3035	0.1648	0.1319	0.1562

Table 1 – Tabel containing the main results of the simulations

5.4 Comparison between MPC and proportional controller

It was also considered a simpler form of MPC (without the usage of time scaling) for the circular trajectory, with the goal of comparing with the performance of the proportional controller based on the invariant error function. Figures from 41 to 52 show the main results obtained for the two methods considered. The reference circular trajectory corresponded to the same path of the one considered for the methods that used time scaling. For both controllers, the obtained trajectory presented reasonable results, as is observed in the three-dimensional plots in Figures 41 and 47 and in the 2D plots shown in Figures 41 and 47. The curves obtained for the rotation axis, illustrated in figures 43 and 49, show a small deviation in relation to the reference for both methods. It is important to highlight that a different error metric was used for the proportional controller. Even so, the obtained errors presented the same order of magnitude, also for the translation, what can be observed in Figures 44 and 50.

As for the joint velocities, whose curves can be seen in Figures 45 and 46 for the Proportional controller and in Figures 51 and 52 for the MPC method. For this trajectory, it was considered, for MPC, bounds of $\pm 2 rad/s$ for the joint velocities and $\pm 40 rad/s^2$ for the joint accelerations. The intention was to show that certain limits can be imposed to the joint variables, as the trajectory is still feasible. The mean error produced by the proportional method was 0.0014, while its maximum reached 0.0097. The mean error for the MPC controller was 0.0018, while its maximum reached 0.0054.

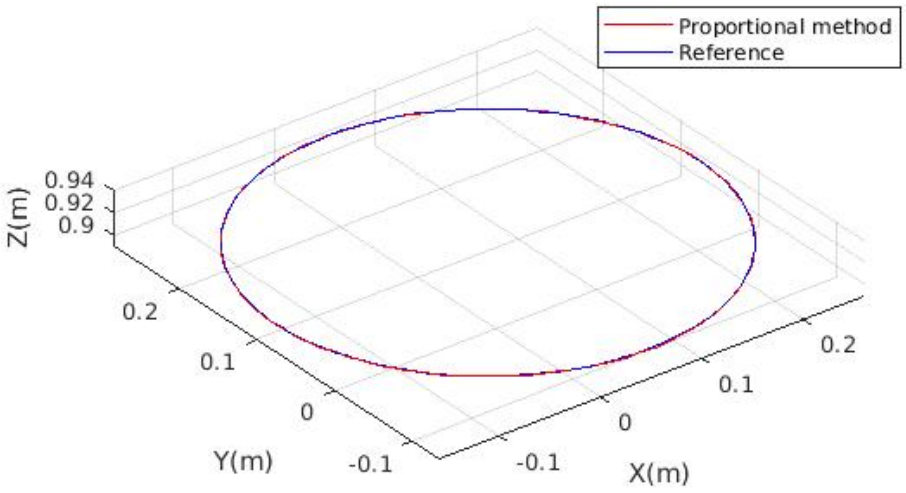

Figure 41 – 3D Plot of circular trajectory using the proportional controller

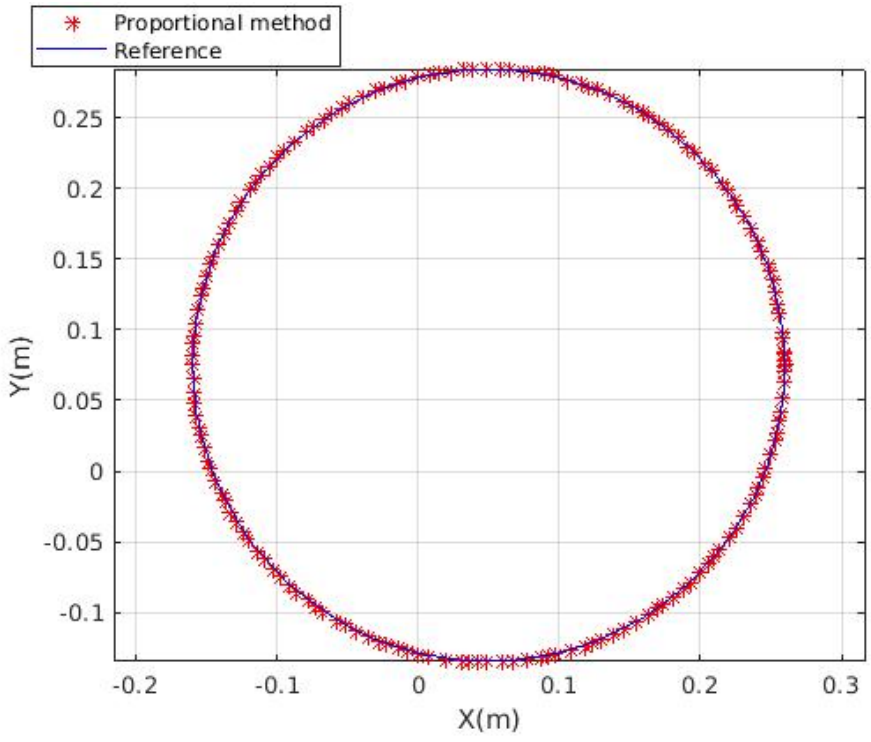

Figure 42 – 2D Plot of circular trajectory using the proportional controller

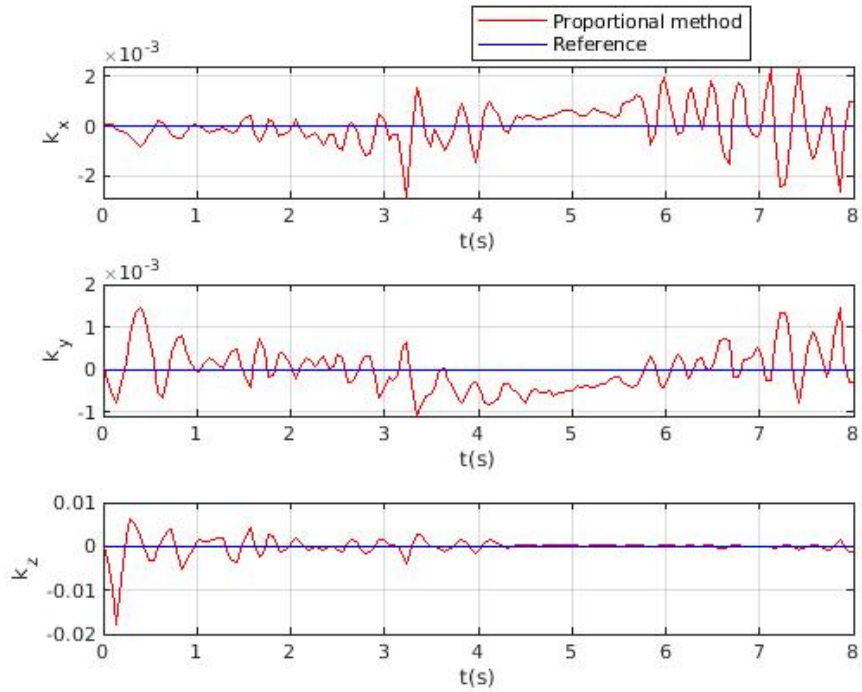

Figure 43 – Curves of each element of the rotation axis, using the proportional controller

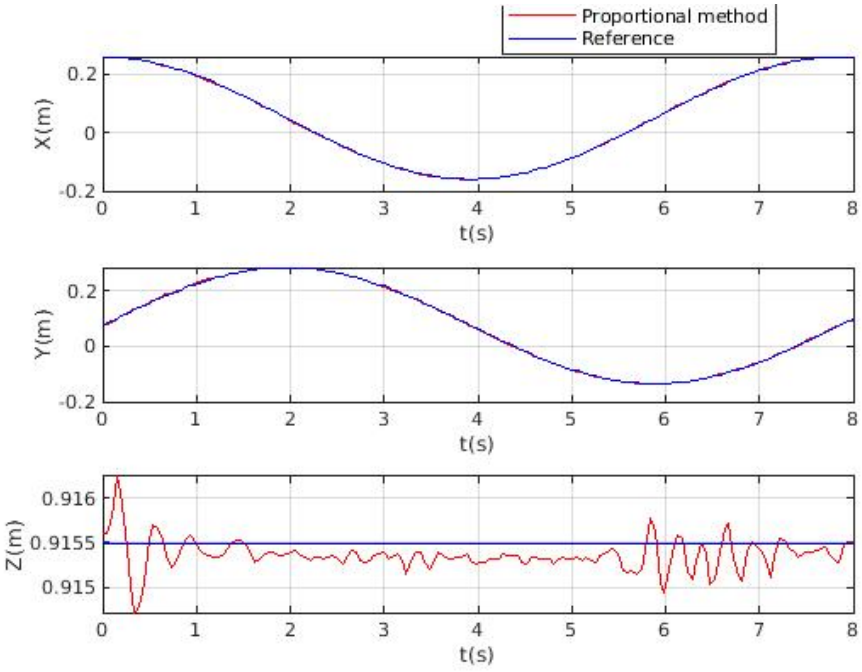

Figure 44 – Curves of each translation element, using the proportional controller

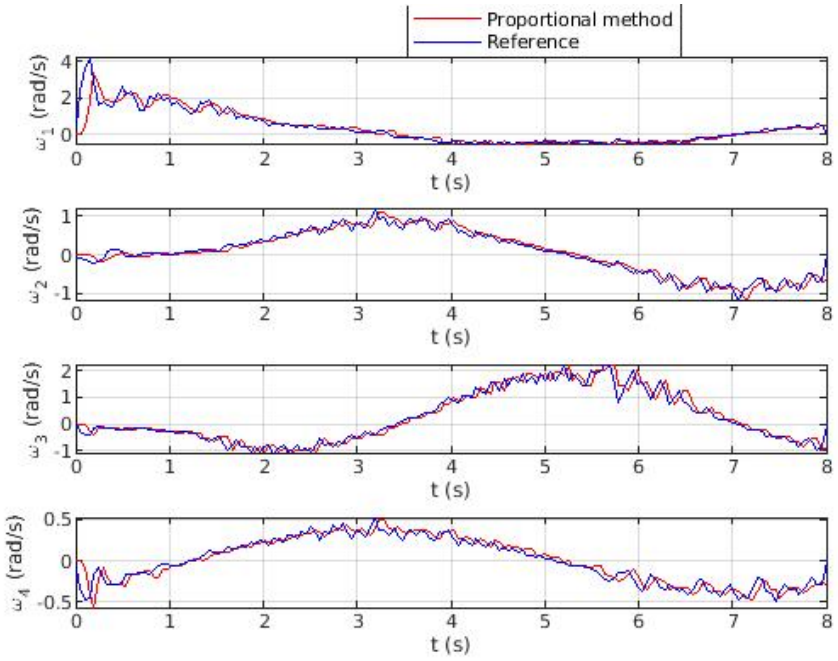

Figure 45 – Curves of joint velocities (from joint 1 to 4), using the proportional controller

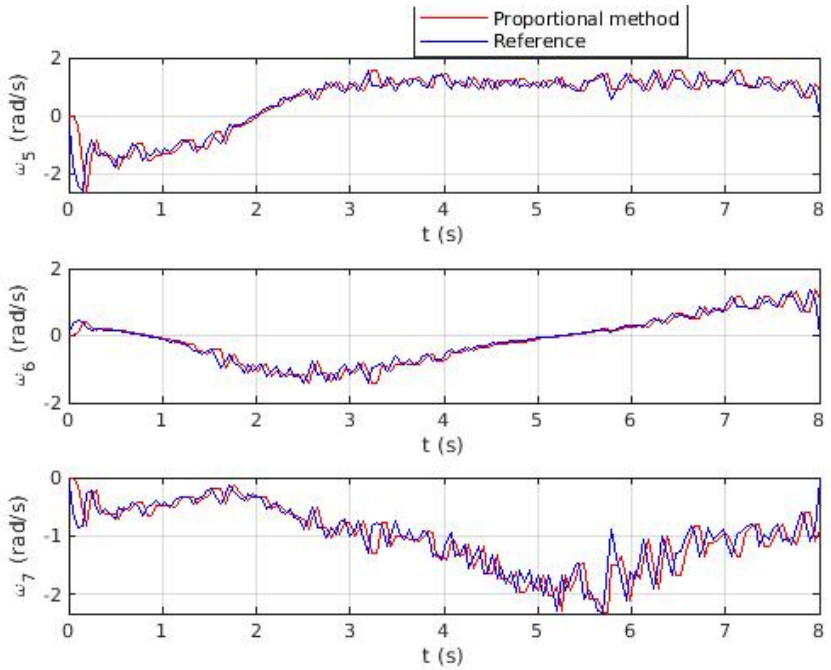

Figure 46 – Curves of joint velocities (from joint 5 to 7), using the proportional controller

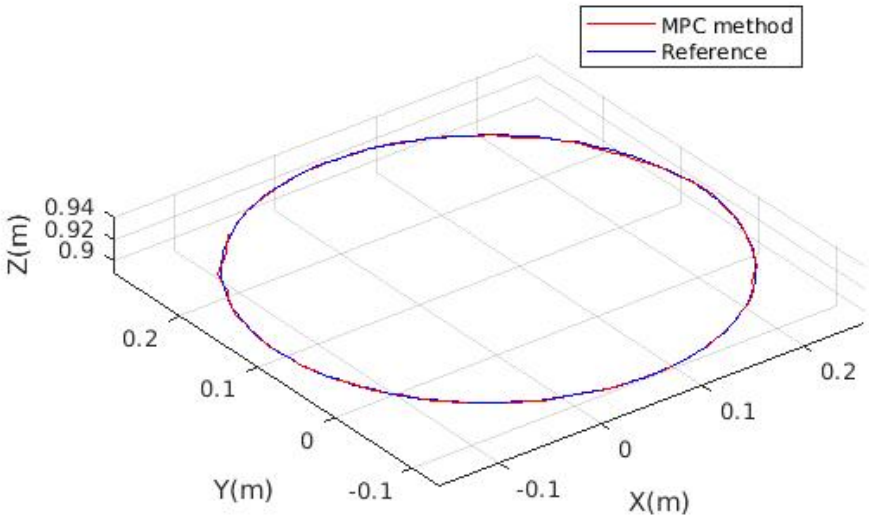

Figure 47 – 3D Plot of circular trajectory using the MPC controller

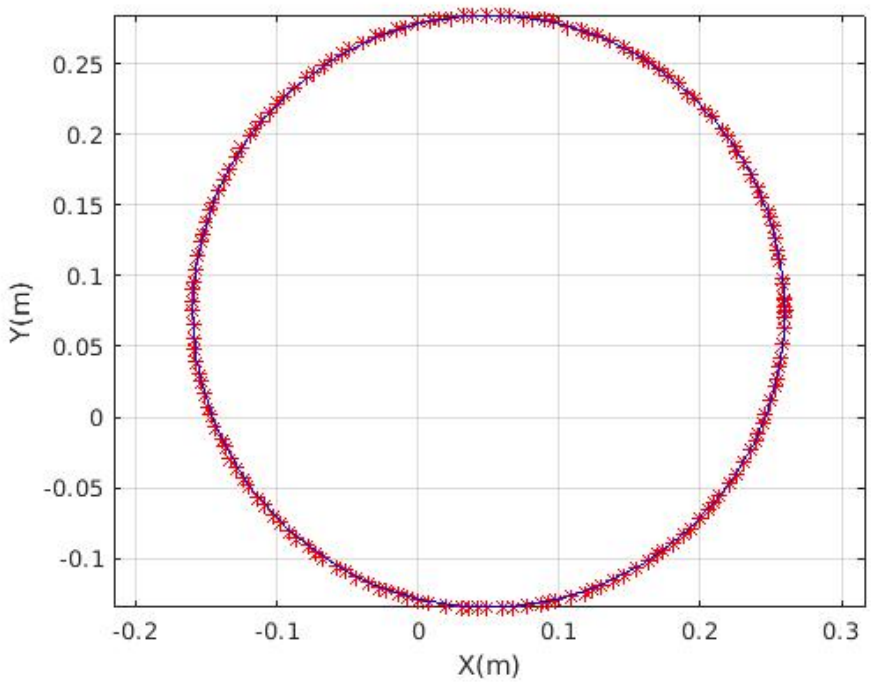

Figure 48 – 2D Plot of circular trajectory using the MPC controller

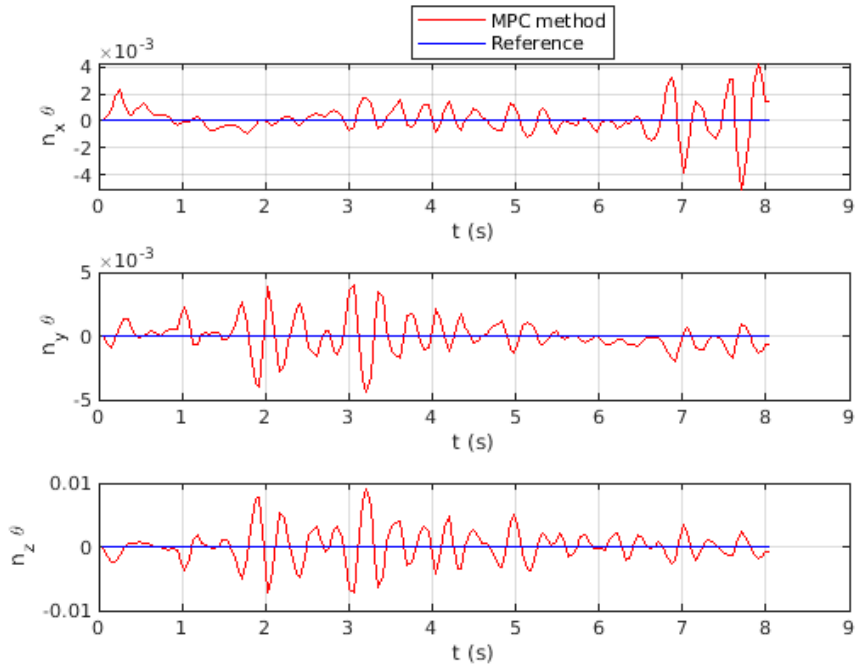

Figure 49 – Curves of each element of the rotation axis, using the MPC controller

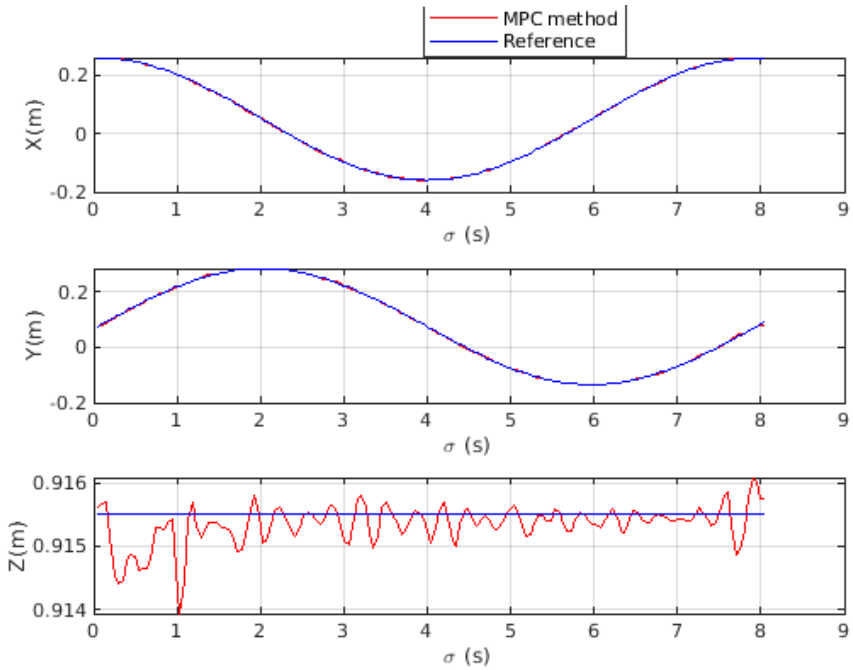

Figure 50 – Curves of each translation element, using the MPC controller

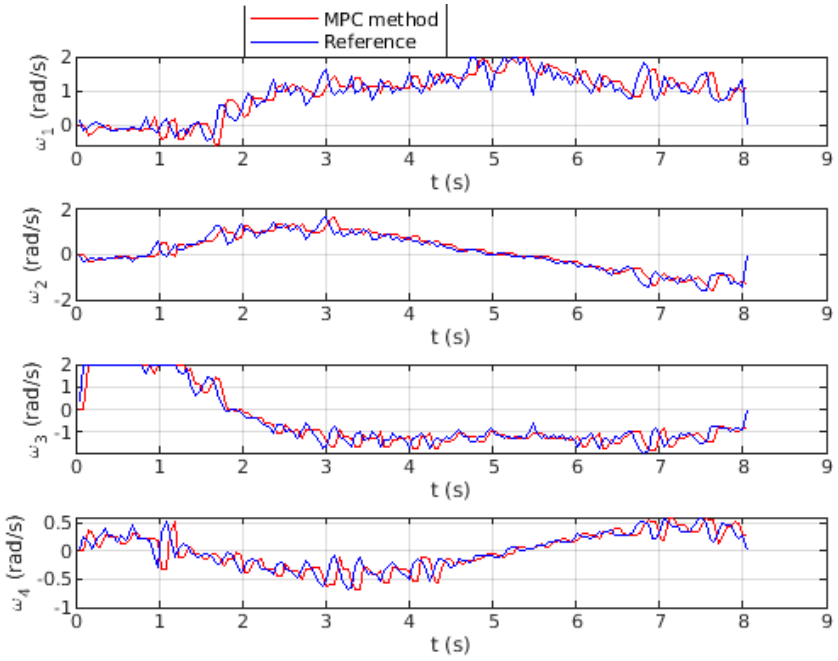

Figure 51 – Curves of joint velocities (from joint 1 to 4), using the MPC controller

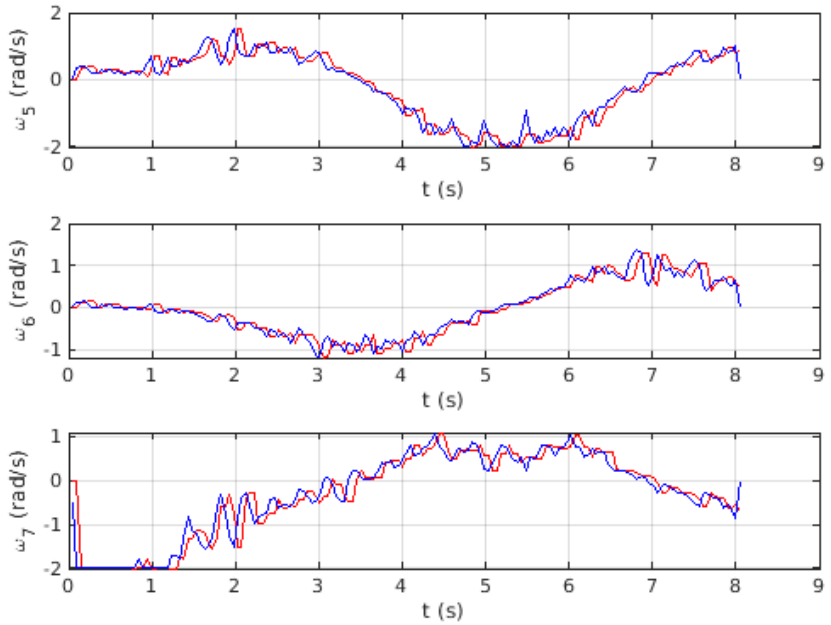

Figure 52 – Curves of joint velocities (from joint 5 to 7), using the MPC controller

6 Conclusions

This project presented the evaluation of two kinds of kinematic controllers using dual quaternions. It was possible to explore a lot of different tools, such as ROS, DQ Robotics, Matlab and the simulator CoppeliaSim. The usage of ROS to exchange data between Matlab and CoppeliaSim has shown to be very convenient. Originally, the node programmed in Lua in the CoppeliaSim scene was made to communicate with a controlling node in C++, but it was very straightforward to adapt the controlling node to Matlab. To work with publishers and subscribers in Matlab was even easier than in C++, in addition to the fact that storing and plotting data using Matlab is much more favorable.

It was clear that the MPC controllers, specially with $p = 10$, was more effective, in general aspects analysed in this project, than the Local method. Although the predictions made for the robot state used an approximation of the Jacobians, they were able to prevent the controller to deform the original path, which has happened for the Local method in both of the trajectories taken into account.

The mean value of the control action also presented better results for the MPC method (specially for the bigger number of predictions). Yet, it does not mean that it necessarily would be obtained lower effort for the joints, as the dynamics were not considered to exert the control of pose. However, for the cases analyzed, the torques applied presented lower peaks for the MPC method.

As for the scaling factor, in the circular trajectory, the MPC method produced much better results (higher mean values), which reinforces the improvements of the predictive control. For the sinusoidal trajectory, however, the scaling factor for the Local method, in all the cases, was higher than the ones obtained through the MPC controller. However, MPC (still for the sinusoidal path) produced lower mean values and did not present deformations in the obtained path such as the Local controller.

A great challenge encountered in this work was to tune a good tuning for the MPC controller: for a specific set of constraints, a certain range of proportional gain and a control horizon worked well. The changing of these different parameters was done several times before it could be obtained good results for the MPC. In fact, it was only possible to achieve significant results after considering constraints between the predictive control actions (which corresponds to a limitation in the mean acceleration).

The comparison between the simpler MPC method and the Proportional controller based on the invariant error function was made with the main goal to show that a feasible trajectory can be made with the two controllers involved, wherein using a QP solver, there is the advantage of imposing some constraints to the control signal.

6.1 Future Works

It would be interesting to continue some features of this work, such as testing other kinds of trajectories, that even involve variant orientation through the path. The trajectories executed in this project considered a null rotation of the end-effector pose along the path and, even so, there was a small noisy error associated to orientation.

A limitation of this work is that it was not formulated a low level control for the dynamics of the joints. The employment of such control level could improve some results and its small disturbances and even a better formulation of the constraints. The study of the dynamics of the manipulator, in general, would be interesting to make more precise elaboration of the restraints.

Another aspect that could improve would be to minimize the different cost functions (of inverse kinematics and scaling) using the lexicographic method, as in this work, the lagrange multipliers λ_1 and λ_2 were applied, which decreases the precision of the resolution.

What could also be studied is the fulfillment of a task that fits in the null space of the primary assignment. This project considered only one geometrical task, but it is possible to consider more than one, which would add a considerable complexity to the work.

Finally, the sampling time achieved in this work was $T = 0.05s$ which is higher than the desired one of the order of milliseconds. That influences the quality of the kinematic control and also the magnitude of the proportional gain applied, so a future work could work on that manner.

References

ADORNO, B. V. Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra. **HAL Archives**, v. 1, Feb. 2017. Cit. on pp. 16–22.

ADORNO, B. V.; MARINHO, M. M. DQ Robotics: A Library for Robot Modeling and Control. **IEEE Robotics & Automation Magazine**, v. 28, p. 102–116, 2021. Cit. on pp. 35, 36.

B. SICILIANO L. SCIAVICCO, L. V.; ORIOLO, G. **Robotics - Modelling, Planning and Control**. Springer, 2009. Cit. on p. 19.

CAMACHO, E.; BORDONS, C.; ALBA, C. **Model Predictive Control**. Springer London, 2004. (Advanced Textbooks in Control and Signal Processing). ISBN 9781852336943. Available from: <https://books.google.com.br/books?id=Sc1H3f3E8CQC>. Cit. on pp. 25, 26.

CRAIG, J. **Introduction to Robotics: Mechanics and Control**. Pearson/Prentice Hall, 2005. (Addison-Wesley series in electrical and computer engineering: control engineering). ISBN 9780201543612. Available from: <https://books.google.com.br/books?id=MqMeAQAAIAAJ>. Cit. on p. 12.

DE ALMEIDA, P. **Indústria 4.0: Princıpios básicos, aplicabilidade e implantação**. Saraiva Educação S.A. ISBN 9788536530468. Available from: <https://books.google.com.br/books?id=cYywDwAAQBAJ>. Cit. on p. 12.

DIMITROV, D.; WIEBER, P.-B.; FERREAU, H. J.; DIEHL, M. On the implementation of model predictive control for on-line walking pattern generation. In: 2008 IEEE International Conference on Robotics and Automation. 2008. P. 2685–2690. DOI: 10.1109/ROBOT.2008.4543617. Cit. on p. 29.

FARONI, M.; BESCHI, M.; PEDROCCHI, N.; VISIOLI, A. Predictive Inverse Kinematics for Redundant Manipulators With Task Scaling and Kinematic Constraints. **IEEE Transactions on Robotics**, v. 35, n. 1, p. 278–285, 2019. DOI: 10.1109/TRO.2018.2871439. Cit. on pp. 26–28, 30.

FARONI, M.; BESCHI, M.; TOSATTI, L. M. A Predictive Approach to Redundancy Resolution for Robot Manipulators. **IFAC (International Federation of Automation and Control)**, 2017. Cit. on pp. 26, 27, 29.

FIGUEREDO, L. F. D. C. **Kinematic Control Based on Dual Quaternion Algebra and its Application to Robot Manipulators**. 2016. PhD thesis – Universidade de Brasília. Cit. on pp. 17–19.

FLOUDAS, C. A.; VISWESWARAN, V. **"Quadratic Optimization" Nonconvex Optimization and Its Applications**. 1995. Cit. on p. 27.

FRANK, M.; WOLFE, P. An algorithm for quadratic programming. **Naval Research Logistics Quarterly**, v. 3, n. 1-2, p. 95–110, 1956. DOI: https://doi.org/10.1002/nav.3800030109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030109. Available from: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030109>. Cit. on p. 27.

HAMILTON, W. R. On Quaternions, Or On a New System Of Imaginaries In Algebra. **Philosofical Magazine**, 1844. Cit. on p. 16.

KUIPERS, J. **Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality**. Princeton University Press, 1999. Cit. on pp. 18, 19.

MARINHO, M. M. **Robot-aided Endoscope Control Under Laparoscopic Surgery Constraints Using Dual Quaternions**. Univerdiade de Brasília, 2014. Cit. on p. 19.

MATLAB. **version 7.10.0 (R2010a)**. Natick, Massachusetts: The MathWorks Inc., 2010. Cit. on p. 15.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. ROS: an open-source Robot Operating System. In: v. 3. Cit. on p. 34.

REIS, F. **Revolução 4.0 A Educação Superior Na Era Dos Robôs**. EDITORA DE CULTURA. ISBN 9788529302126. Available from: <https://books.google.com.br/books?id=hNQSygEACAAJ>. Cit. on p. 12.

ROHMER, E.; SINGH, S. P. N.; FREESE, M. V-REP: A versatile and scalable robot simulation framework. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2013. P. 1321–1326. DOI: 10.1109/IROS.2013.6696520. Cit. on p. 14.

ROHMER, E.; SINGH, S.; FREESE, M. V-REP: A versatile and scalable robot simulation framework. In: p. 1321–1326. DOI: 10.1109/IROS.2013.6696520. Cit. on p. 32.

SANTOS, M.; LEME, M.; STEVAN, S. **Indústria 4.0: FUNDAMENTOS, PERSPECTIVAS E APLICAÇÕES**. ERICA, 2018. ISBN 9788536527208. Available from: <https://books.google.com.br/books?id=v7yStwEACAAJ>. Cit. on p. 12.

SELIG, J. M. **Geometric Fundamentals of Robotics**. 2. ed.: Springer-Verlag New York Inc., 2005. Cit. on p. 22.

SILVA PEREIRA, M. da. **Trajectory Control of Anthropomorphic Compliant Manipulator with Dual Quaternion Based Kinematic Controllers**. Univerdiade de Brasília, 2016. Cit. on pp. 19, 22, 23, 32.

SPONG, M.; HUTCHINSON, S.; VIDYASAGAR, M. **Robot Modeling and Control**. Wiley, 2005. ISBN 9780471649908. Available from: <https://books.google.com.br/books?id=jyD3xQEACAAJ>. Cit. on p. 32.

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL. **Robotic Operating System**. 23 May 2018. Available from: <https://www.ros.org>. Cit. on p. 15.

WANG, L. Model Predictive Control System Design and Implementation Using MATLAB. In. Cit. on p. 25.

YOONSEOK PYO HANCHEOL CHO, R. J.; LIM, T. **ROS Robot Programming**. ROBOTIS Co.,Ltd., 2017. ISBN 979-11-962307-1-5. Available from: <www.robotis.com>. Cit. on pp. 34, 35.