Technical Report

IRI-TR-11-01

Smooth Inverse Kinematics Algorithms for Serial Redundant Robots

Master Thesis

Adrià Colomé

September, 2011

Institut de Robòtica i Informàtica Industrial

Abstract

The inverse kinematics of a robot is the mapping that, given a goal position, calculates a set of joint positions so as to place the robot's end effector in the specified goal. As this is a relevant issue to move the robot, there has been a lot of work about obtaining a fast and robust inverse kinematic algorithm. In this work we present the main concerns on finding an inverse kinematics algorithm for a serial redundant manipulator.

We firstly comment on our motivation, the state of the art and set our objectives, to later briefly expose some necessary concepts. Then the work is divided into two parts: the first one, describing analytical methods for solving inverse kinematics, and the second one about control-based methods.

In the analytical methods, we will see how difficult it may be to have a closed analytical form, in particular for redundant manipulators, and we will try to solve, not always with success, the inverse kinematics of some robots.

Then we will move into control-based algorithms, which are iterative methods using the Jacobian matrix of a robot, and we will firstly discuss the concerns of these methods, which mainly are:

- Convergence: Achieving the specified goal from a given starting position.
- Joint limit avoidance: Given the valid interval for each joint, the algorithm should not give a solution which cannot be reached due to the joints physical limitations.
- Robustness: When a singularity occurs, the algorithm must be capable of not getting stuck or become chaotic.
- Cost: The time it takes to find a solution.

We will present a survey on the most used existing control-based algorithms. We point out the drawbacks of each one of them, to later propose a new way of numerically filtering the Jacobian matrix, and also a new method to avoid step-depending convergence issues, while respecting joint limits, to have a very smooth and robust algorithm. All these methods have been implemented using *Matlab* to solve the inverse kinematics of some robots. The results of two of them, a 4R planar robot and the Barrett's WAM arm, are shown so as to draw conclusions.

To end up, we discuss the convergence of these methods, and the global use of them. The algorithm we propose has the advantadge of successfully respecting joint limits and its convergence is not being step-dependent makes it more robust. Nevertheless, we also discuss the need of, when the goal is far from the starting position, add path planning methods to approach the target.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

Corresponding author:
Adrià Colomé
tel: +34 93 401 5791
acolome@iri.upc.edu
http://www.iri.upc.edu/staff/acolome

Contents

1 Preliminaries

1.1 Motivation

Year after year, more complex serial robots are designed and built. Many of them with multiple limbs (arms / fingers) and redundant, in the sense that they are *overarticulated*. This overarticulation gives more flexibility, and now robots can not only reach a position, but also reach it in several configurations, and so secondary goals can be achieved.

Nevertheless, this complexity on the robots has a drawback, which is the computation of their kinematics. The kinematics map of a serial robot is not a bijection. In particular, the kinematics of a serial robots is a *surjective* function with its reachable workspace (i.e.: a mapping $f : X \to Y$ so that $\forall\, y \in Y$, there exists one or more $x \in X$ so that $f(x) = y$), in the sense that two different joint states can give the same position of the *end effector* of a robot.

The initial aim of this project has been to find an efficient, continuous solution for the inverse kinematics of the *Barrett's WAM arm*, a robot available in the *Institut de Robòtica i Informàtica Industrial (IRI)*, whose of-the-shelf inverse kinematics series algorithm is not optimum. This aim has been focused on analysing its inverse kinematics by analytical methods, but also with control-based methods, to compare their solutions.

1.2 State of the art

Inverse kinematics algorithms have been an issue to focus on since the first robots were built. To do so, the most popular methods have been the analytical (and so exact) ones. Nevertheless, an exact solution does not always exist, and when one has to look for an alternative solution, the most popular methods are the control-based ones, although there also exist methods based on learning, neural networks [22], interval methods [16], or based on distances [15].

On control-based methods, the first approaches were developed in the 60s-70s, when Whitney [23] used the pseudoinverse of a matrix to solve the problem. Although other methods such as the Jacobian transpose [24], which can be faster, were used as well. Lately in the 1980s, with the rising computation capability of computers, Baillieul [1] presented an *extended Jacobian*, which was computationally more complex, with second-order derivatives. Also in the 80s, the *manipulability index* was defined by Yoshikawa [25], a term to optimize when having a redundant robot.

Since then, many ways of improving the performance of these algorithms have been proposed. Siciliano and Slotine presented in 1991 a hierarchical way to priorisize different tasks [19], and parallelly, the damping of the Jacobian, a way of avoiding the discontinuity of the pseudoinverse operator when the Jacobian matrix of a robot looses rank, was presented and analised [6] with its variants [5]. In the 90s is when it began to be also very common to weight the Jacobian matrix of a manipulator in order to prioritisize joints or tasks over others [3], or even block certain movements with these weighting matrices.

But it has not been until the 2000s that the *continuity* of these methods started having a relevant paper in the design of algorithms. In [17], Raunhart and Boulic try to continuously block the motion of a joint, and finally in [13], Mansard, Remazeilles and Chaumette define a new pseudoinverse operator to be smooth with respect to the joint/task blocking, to be used later in [12], combining this continuous pseudoinverse operator with a task priority scheme.

1.3 Objectives

As commented on the motivation, our objective is to find an analytical solution for the inverse kinematics of redundant robots, and regarding control-based methods, to obtain the smoothest algorithm as possible, leading to a robust iterative method.

To do so, our intention is, after considering the analytical ways of solving the problem, to describe the most popular existing control-based methods, trying to point out their limitations and difficulties, in a survey-like way, to then try to take the best of each of them and find a robust solution, checking its behaviour and comparing it to the analytical one.

2 Introduction

Controlling the position of a robot is not as trivial as one could think. Serial robots are composed of concatenated joints, forming an open chain structure. With this structure, the inverse kinematic problem, that is, relating the position and/or orientation of the *end effector* of the manipulator with the position of each joint, might not have a closed analytical solution. Not having such an analytical solution can force the user to search for alternative solutions to the kinematics of the robot. And even when a closed analytical form exists, its equations may have multiple solutions.

Ever since the first industrial robots appeared, there has been a big concern about computing kinematics in a simple, reliable and fast way in order to optimize the robot performance. In this work, we analyse the state of the art on the computation of inverse kinematics, pointing out the weaknesses of all the methodologies considered, and aiming to take the best of them to obtain an inverse kinematics algorithm which is robust and reliable, with special attention to redundant robots (those with more degrees of freedom than supposely needed to perform a specified task).

2.1 Forward and inverse kinematics of a serial manipulator

Let us consider a serial robot with m joints, and let $W \subseteq \mathbb{R}^n$ be its *workspace* (reachable positions) in a space of dimension n (n=*2* for planar position, 3 for planar position+orientation or spatial position, 6 for spatial position+orientation, etc.). Then we define:

- **The Forward Kinematics** as the function that maps a joint position of the set Q of feasible joint positions into a cartesian position in the workspace W

$$\begin{aligned} f : Q \subseteq \mathbb{R}^m &\rightarrow W \subseteq \mathbb{R}^n \\ q &\mapsto x \end{aligned}$$

  The forward kinematics function is easily obtained for serial manipulators with the *Denavit-Hartenberg* coefficients (see section 4.4.3). By construction, $W = Im_Q(f)$

- **The Inverse Kinematics** is the inverse mapping of the Forward kinematics, i.e.: the function that maps a position in the workspace to a joint position:

$$\begin{aligned} h : W \subseteq \mathbb{R}^n &\rightarrow Q \subseteq \mathbb{R}^m \\ x &\mapsto q \end{aligned}$$

  Note that h can have multiple solutions for a single x, or even infinite solutions in case $m > n$ or in a *degenerate position*. Finding a closed expression for h can be very complicated or even impossible, and when handling with trigonometric functions, more than one solution may arise.

2.2 Denavit-Hartenberg parameters

When considering a serial open-chain manipulator, a systematic way of expressing its forward kinematics is using the *Denavit-Hartenberg* convention. This consists in defining the relative position and orientation of two consecutive links, and determining a transformation matrix between them, by performing the following operations (see Figure 1):

First, consider axis z_{i-1} as the axis of the joint between link *i-1* and link i. To define the homogeneous transformation between frames *i-1* and i:

- Set axis z_i as the axis of rotation or movement of joint *i+1*.

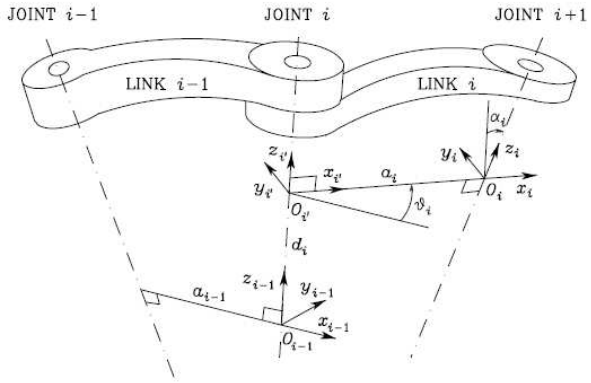

Figure 1: Denavit-Hartenberg parameters calculation

- Find the common normal line s between z_{i-1} and z_i, Define the origin of frame i in $O_i = s \bigcap z_i$, and also $O'_i = s \bigcap z_{i-1}$
- Define the axis x_i in the direction of s, from joint i to joint *i+1*.
- $y_i = z_i \times x_i$, defined following the right-hand rule.

Now, we define:

- a_i is the distance (with sign) from O'_i to O_i along axis x_i.
- d_i is the distance (with sign) from O_{i-1} to O'_i along axis z_{i-1}.
- α_i is the angle (with sign following the right-hand rule) from axis z_{i-1} to z_i around x_i
- θ_i is the angle (also with sign following the right-hand rule) from axis x_{i-1} to x_i around z_{i-1}

The parameters α_i and a_i are always constant, while θ is a variable if joint i is rotational and d is a variable if joint i is prismatic.

Also, one must consider some ambiguity situations:

- The first frame has only z_0 direction defined, so O_0 and x_0 can be chosen arbitrarily.
- For the last frame *(n)*, as no joint *n+1* exists, z_n is not uniquely defined and can also be chosen.
- When two consecutive axes intersect, x_i is arbitrarily chosen.
- When two consecutive axis are parallel, z_i can be placed along a set of parallel lines.
- When joint i is prismatic, then the parameter d calculated is taken as an offset on the variable d_i.
- When joint i is revolute, then the parameter θ calculated is taken as an offset on variable θ_i.

With these parameters and considerations, the homogeneous transformation from frame *i-1* to frame i is:

$$T_i^{i-1} = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) \cdot cos(\alpha_i) & sin(\theta_i) \cdot sin(\alpha_i) & a_i \cdot cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \cdot cos(\alpha_i) & -cos(\theta_i) \cdot sin(\alpha_i) & a_i \cdot sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

And for a several joints manipulator, the homogeneous transformation of the end effector has the form: $T_{ee}^b = T_0^b \cdot T_1^0 \cdot ... \cdot T_m^{m-1} \cdot T_{ee}^m$, where T_0^b is the homogeneous transformation between a base frame and the joint 1 frame. T_{ee}^m is the tool end effector relative position to the end joint.

3 Robot positioning

Positioning a robot in its workspace requires calculating its inverse kinematics. In this section, we introduce the basic concepts for understanding the following sections in a very practical and focused way.

3.1 Robot differential kinematics

There is much literature about differential kinematics, in terms of joints and position velocity and accelerations, etc. But here we will focus on what will be used later. For further reading, we recommend [20], chapter 3.

Given a serial manipulator, one might want not only to know its direct and inverse kinematics, but also its differential kinematics. In this sense, a fundamental tool is the *Geometric Jacobian* of a manipulator at a given joint state θ, which gives the differential position/orientation first-order variation of the end effector of a manipulator, depending on the differential first-order variation of each joint, i.e.:

$$\dot{x} = J \cdot \dot{\theta}, \tag{1}$$

where $\dot{\theta}$ is the time-derivative of the joints, and x is the generalized coordinate vector in the cartesian space:

$$x = \begin{pmatrix} x_e \\ \phi_e \end{pmatrix},$$

ϕ_e being a set of Euler angles describing the orientation of the end effector.

Column i of the Jacobian J represents the effect of ith joint on the generalised coordinates of the end effector.

To compute this Jacobian in a m-link robot, this formula can be applied (see [20], pp 111-113):

$$J = \begin{bmatrix} J_{P1} & ... & J_{Pm} \\ J_{O1} & ... & J_{Om} \end{bmatrix},$$

where:

$$\begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

if the joint is prismatic, and

$$\begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{bmatrix} z_{i-1} \times (P_e - P_{i-1}) \\ z_{i-1} \end{bmatrix}$$

if the joint is revolute,

where P_e is the position of the end effector of the robot and P_{i-1} is the position vector of the origin of frame *i-1*, both expressed in the base frame.

3.2 Desired position and error representation

Depending on the type of robot and the task to be performed, the state vector may vary from a *2*-dimentional position (x, y) to a generalized 6-dimensional position and orientation vector. In any case, a planar position is a projection of the generalised coordinate vector.

Now, to express the orientation error, given a desired position and/or orientation, if we take the non-minimal representation of the actual orientation as the rotation matrix

$$R_e(\theta) = \begin{pmatrix} n_e(\theta) & s_e(\theta) & a_e(\theta) \end{pmatrix},$$

and equally with the desired orientation:

$$R_d = \begin{pmatrix} n_d & s_d & a_d \end{pmatrix},$$

then the orientation error can be represented as (see [20], pp 137-140):

$$e_o = \frac{1}{2}(n_e(\theta) \times n_d + s_e(\theta) \times s_d + a_e(\theta) \times a_d), \tag{2}$$

while the position error is:

$$e_P = x_d - x_e(\theta) \tag{3}$$

This error representation will be useful to perform control-based methods for inverse kinematics.

3.3 Criteria for optimizing redundant degrees of freedom

When dealing with redundant manipulators, as the robot has more degrees of freedom than necessary to perform a certain task, the remaining degrees of freedom give a set of feasible solutions of the inverse kinematics. Among these solutions, it is recommended to choose the one satisfying a certain criterion: A secondary goal, an additional restriction, etc. For almost every possible task, a potential function can be created; if $H : \mathbb{R}^m \to \mathbb{R}$ is a cost function, a gradient of the form $\vec{h} = - \bigtriangledown H$ can be used. Now we will describe some criteria that are often used.

3.3.1 Manipulability

The manipulability of a robot in a given configuration θ was first introduced by Yoshikawa [25], and it is defined as: $w(q) = \sqrt{|det(J(\theta) \cdot J(\theta)^T)|}$.

With a singular value decomposition, and using $det(AB) = det(A)det(B)$, it can be easily proven that $w(q) = \prod \sigma_i$, the product of the eigenvectors of J, and now we will see some properties of interest of this term:

Consider the set of joint velocities of constant unit norm

$$\dot{\theta}^T \cdot \dot{\theta} = 1. \tag{4}$$

Assuming full-rank matrix J, we can set a minimum norm solution of the inverse kinematics from (1)

$$\dot{\theta} = J^\dagger \cdot \dot{x}, \tag{5}$$

where $J^\dagger = J^T \cdot (J \cdot J^T)^{-1}$ is the *Moore-Penrose* pseudoinverse matrix of J. This set of velocities maps, through equations (4) and (5), to a corresponding set of cartesian space velocities (using pseudoinverse properties and assuming full-row rank of J):

$$v_e^T \cdot (J^{\dagger T} \cdot J^\dagger) \cdot v_e = v_e^T \cdot (J \cdot J^T)^{-1} \cdot v_e = 1,$$

which is the equation of the points on the surface of an ellipsoid in the end effector velocity space. The eigenvectors of matrix $J \cdot J^T$ give the principal axes of the ellipsoid, and its eigenvalues give the width of the ellispoid along the corresponding eigenvectors. This ellipsoid gives an idea

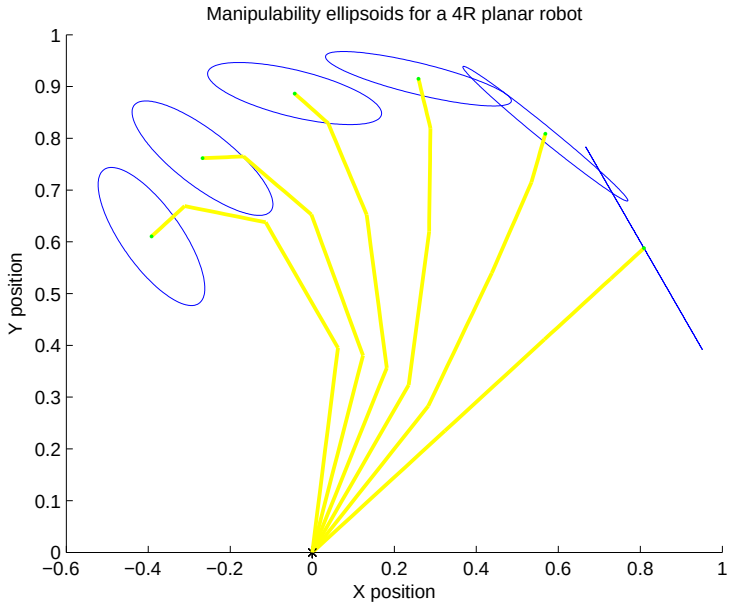

Figure 2: Manipulability ellipsoids (scaled to fit in the image). Here it can be seen that, when approaching a singularity, the task Jacobian looses rank, and the manipulability ellipsoid's area becomes zero.

of the behaviour of the end effector in the neighbourhood of the actual state, and its volume is proportional to $w(\theta) = \sqrt{|det(J(\theta) \cdot J(\theta)^T)|}$. What has to be considered with manipulability is the fact that if one eigenvalue of $J \cdot J^T$ is zero, so is the manipulability index, meaning that if J is rank-deficient, the robot is in a singular position. Also, the higher the manipulability, the less excentricity the manipulability ellipsoid will have, the farther the robot from a singularity will be, and the more isotropic the movement capabilities of the robot are in that position.

In Figure 2 we can see how the manipulability ellipsoid varies with position for a 4R planar robot, and has 0 area at a singularity.

The convenience of using this criterion has been longly discussed, and a similar term as a *manipulability polytope* (see [10]) may reflect the robot's behaviour better, but it is computationally slower.

3.3.2 Joint limit avoidance

Another criterion used in inverse kinematics algorithms is to avoid joint limits. This can be done by optimizing a potential function with very high values in the neigbourhood of a limit. For example, as used in some literature:

$$w(q) = \frac{1}{2m} \sum_{i=1}^{m} \frac{q_i^{max} - q_i^{min}}{(q_i^{max} - q_i)(q_i - q_i^{min})} \tag{6}$$

This function gives, as we can see in Figure 3, a very high potential when approaching a joint limit, and a minimum value at the mid point.

3.3.3 Other Criteria

More criteria can be used in an optimization, for instance, distance to an obstacle, defined as $H(q) = min_{p,o}\|p(q) - o\|$, where p is a point on the robot and o is a point on the obstacle.

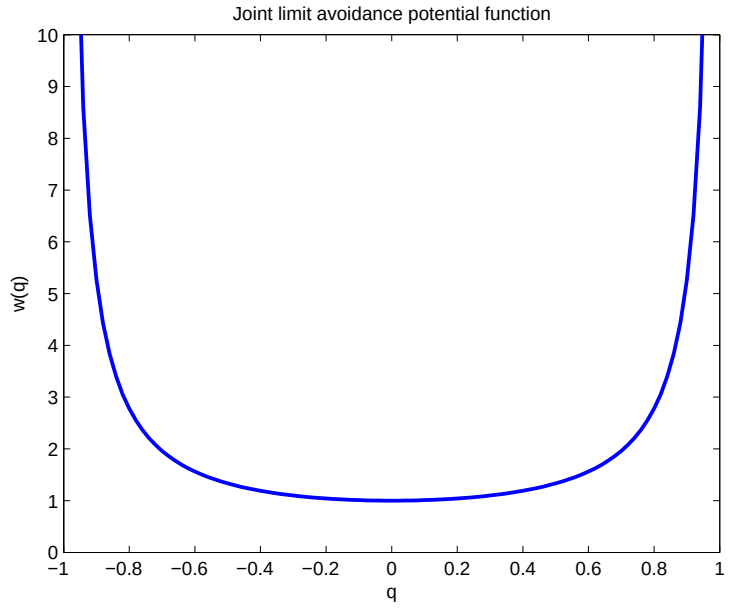

Figure 3: The potential function rises when approaching joint limits.

Another example would be, for a mobile robot, to keep the vertical projection of its center of mass inside a region, which could be the polygon defined by the feet in contact with the ground (legged robot), or its wheels (wheeled robot) , to mantain its equilibrium.

4 Analytical methods for inverse kinematics

When trying to compute inverse kinematics of a manipulator, a recommended first approach is to compute an analytical solution, as it will be an exact solution, and usually faster to compute than any other.

4.1 General procedure

As for a serial robot direct kinematics equations are often easy to obtain (in complex serial robots, the use of the D-H parameters might be useful, although other methodologies exist), a way to start is trying to find the inverse function of the direct kinematics. As many trigonometric functions may appear when dealing with rotational joints, this inversion can require some algebraic skills, as it will usually be non-trivial, or even impossible. Moreover, multiple solutions of trigonometric functions must be considered. This derives in multiple solutions of the inverse kinematics, called *assembly modes*.

Now we will show an example of how difficult this operation can be for a simple manipulator as a 3R (a robot with 3 chained rotational joints as shown in Figure 4).

4.1.1 3R manipulator

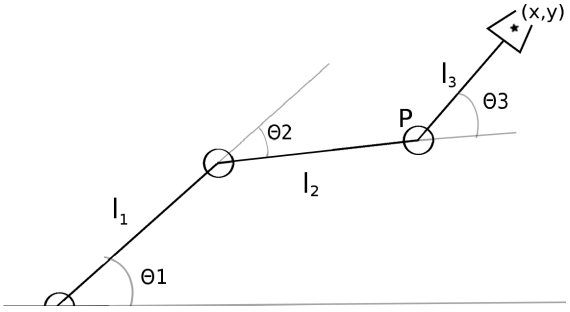

Figure 4: 3R serial robot

By analysing a bit the geometry of the robot, one can easily extract its direct kinematics:

$$\begin{aligned} x &= l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) + l_3 cos(\theta_1 + \theta_2 + \theta_3) \\ y &= l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) + l_3 sin(\theta_1 + \theta_2 + \theta_3) \\ \phi &= \theta_1 + \theta_2 + \theta_3, \end{aligned} \tag{7}$$

In fact, we can compute point P as $P = (x - l_3 cos(\phi), y - l_3 sin(\phi))$ and so we have

$$\begin{aligned} P_x &= l_1 cos(\theta_1) + l_2 cos(\theta_1 + \theta_2) \\ P_y &= l_1 sin(\theta_1) + l_2 sin(\theta_1 + \theta_2) \end{aligned} \tag{8}$$

Using $cos(\theta_1 + \theta_2) = cos(\theta_1)cos(\theta_2) - sin(\theta_1)sin(\theta_2)$ and $sin(\theta_1 + \theta_2) = sin(\theta_1)cos(\theta_2) + cos(\theta_1)sin(\theta_2)$, we have $P_x^2 + P_y^2 = l_1^2 + l_2^2 + 2l_1 l_2 cos(\theta_2) \Rightarrow cos(\theta_2) = \frac{P_x^2 + P_y^2 - l_1^2 - l_2^2}{2l_1 l_2}$. For the sine of θ_2, we can use $sin(\theta_2) = \pm\sqrt{1 - cos(\theta_2)}$, and so calculate the sign-sensible arc tangent: $\theta_2 = atan2(sin(\theta_2), cos(\theta_2))$ for the two solutions of θ_2

Now, given the value of θ_2, and inserting it in equation (8), we get the following linear system:

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \begin{pmatrix} l_1 + l_2 cos(\theta_2) & -l_2 sin(\theta_2) \\ l_2 cos(\theta_2) & l_1 + l_2 cos(\theta_2) \end{pmatrix} \cdot \begin{pmatrix} cos(\theta_1) \\ sin(\theta_1) \end{pmatrix},$$

which can be solved to obtain $cos(\theta_1)$, $sin(\theta_1)$ and so θ_1. Note that these equations have two solutions for θ_1, so two sets of solutions $\theta_1, \theta_2, \theta_3$ are obtained. These two solutions correspond to different *assembly modes*, which are different joint configurations for a non-redundant situation that give the same end effector position. In some ways, it can be seen as inner/outer elbow situation, as we can see in Figure 5.

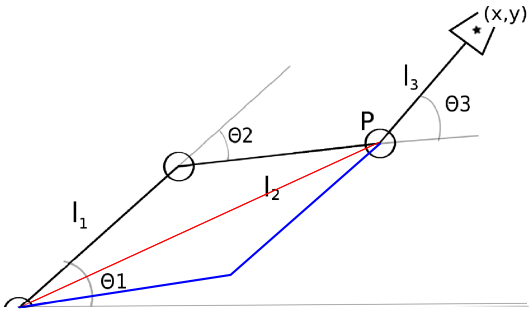

Figure 5: 3R serial robot with two assembly modes

Once θ_1 and θ_2 are known, it is inmediate to obtain $\theta_3 = \phi - \theta_1 + \theta_2$.

As we have seen, it is not straightforward to obtain the inverse kinematics of a simple 3-joint manipulator. This complexity grows with the complexity of the robot geometry, and it is very useful to split the problem into several ones when possible. One such case of this situation is when having a spherical wrist.

4.2 Spherical Wrists

A typical architecture for spatial robotic arms is that the robots last degrees of freedom form a so-called *spherical wrist* (see Figure 6),

link	a_i	α_i	d_i	θ_i
i	0	$-\pi/2$	d_i	θ_i
i+1	0	$\pi/2$	0	$\theta_i + 1$
i+2	0	0	d_{i+2}	$\theta_i + 2$

With this set of DH-parameters, the forward kinematics can be expressed as:

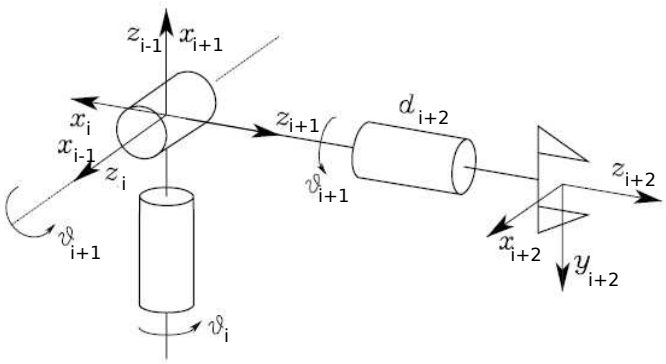

Figure 6: Spherical wrist. Picture from [20].

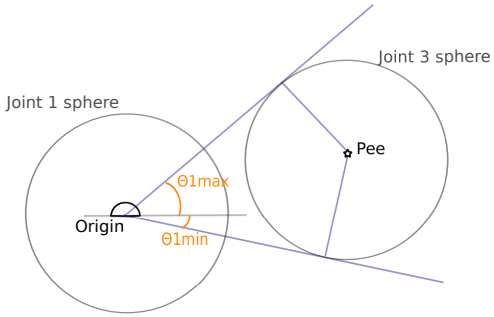

Figure 7: Maximum and minimum angle θ_1 in a 3R manipulator

$$T_{i+2}^{i-1} = T_i^{i-1} \cdot T_{i+1}^{i} \cdot T_{i+2}^{i+1} = \begin{pmatrix} c_i c_{i+1} c_{i+2} - s_i s_{i+2} & -c_i c_{i+1} s_{i+1} - s_i c_{i+2} & c_i s_{i+1} & c_i s_{i+1} d_{i+2} \\ s_i c_{i+1} c_{i+2} - c_i s_{i+2} & -s_i c_{i+1} s_{i+1} - c_i c_{i+2} & s_i s_{i+1} & s_i s_{i+1} d_{i+2} \\ -s_{i+1} c_{i+2} & s_{i+1} s_{i+2} & c_{i+1} & c_{i+1} d_{i+2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

With this transformation, supposing a robot of m degrees of freedom, the last 3 of them corresponding to a spherical wrist. Then, given an objective homogeneous transformation for its end effector, T_{ee}^0, we have $T_{ee}^0 = T_{m-3}^0 \cdot T_m^{m-3} \cdot T_{ee}^m$, where $T_m^{m-3} = T_{SW}$ is the spherical wrist homogeneous transformation. Then: $T_m^0 = T_{m-3}^0 \cdot T_{SW} \Rightarrow (T_{m-3}^0)^{-1} \cdot T_m^0 = T_{SW}$.

Then, when having a robot ending in a spherical wrist, one can compute the wrist point $P_w = P_e - d_{i+2} z_e$, where P_e is the end effector point and z_e is the desired end effector z-axis orientation. Then compute the inverse kinematics of the rest of the robot to reach the position P_w and, when done, take $R_{SW} = \begin{pmatrix} \mathbf{n} & \mathbf{s} & \mathbf{a} \end{pmatrix}$, and the spherical wrist solutions are:

$$\begin{aligned} \theta_4 &= atan2(\epsilon a_y, \epsilon a_x) \\ \theta_5 &= atan2(\epsilon \sqrt{a_x^2 + a_y^2}, a_z) \\ \theta_6 &= atan2(\epsilon s_z, -\epsilon n_z), \end{aligned}$$

where $\epsilon = \pm 1$ (a spherical wrist has two sets of solutions).

4.3 Redundant degrees of freedom optimization

Sometimes the robot has more degrees of freedom than supposely needed for achieving a task.In [18] and [21], it is proposed for some robots to find a parametric expression of the redundant degrees of freedom to perform optimization on the parameters.

4.4 Examples

Now we will show how to compute analytically the inverse kinematics of three redundant robots, to show how difficult it might be.

4.4.1 3R redundant planar manipulator

First, we will find a solution for the robot of Figure 4 assuming that the task is only to set the end effector in a 2-dimensional position (x, y) without concerning the orientation. To do so, we propose a geometric solution.

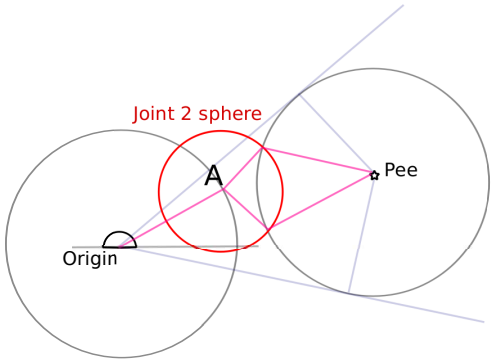

Figure 8: Graphical solution of inverse kinematics

First, given *Pee*, the end effector desired position, and the 3 links lengths, we can find the maximum and minimum values of θ_1 as seen in Figure 7.

With these values, what can be done is to choose an arbitrary value of θ_1, and obtain point A as in (8):

$$A = \begin{pmatrix} l_1 cos(\theta_1) \\ l_1 sin(\theta_1) \end{pmatrix},$$

and then:

$$A + \begin{pmatrix} l_2 cos(\theta_2) \\ l_2 sin(\theta_2) \end{pmatrix} = P_{ee} - \begin{pmatrix} l_3 cos(\theta_3) \\ l_3 sin(\theta_3) \end{pmatrix},$$

so

$$\begin{pmatrix} l_1 cos(\theta_1) + l_2 cos(\theta_2) \\ l_1 sin(\theta_1) + l_2 sin(\theta_2) \end{pmatrix} = \begin{pmatrix} P_{ee}^X - l_2 cos(\theta_2) \\ P_{ee}^Y - l_2 sin(\theta_2) \end{pmatrix}$$

which is a system with 2 equations in 2 variables that can be solved to get θ_2 and θ_3 (with 2 solutions for each θ_1).

With this method, an optimization can be performed to improve a cost function on the parameter θ_1.

4.4.2 Laparoscopic robot

Now an example of a laparoscopic robot will be presented. These robots are used to perform laparoscopic surgery and usually have up to 4 arms to work, but here we will focus on only one arm robot as shown in Figure 9.

In every laparoscopic surgery, there is an *entrance point* P, with a trocar, to insert the robot. The link between points P and *ee* must enter through that point. The patient is then inflated with the end effectors in it, to generate enough workspace to move the tools.

According to the *Denavit-Hartenberg* convention, the robot has the following parameters:

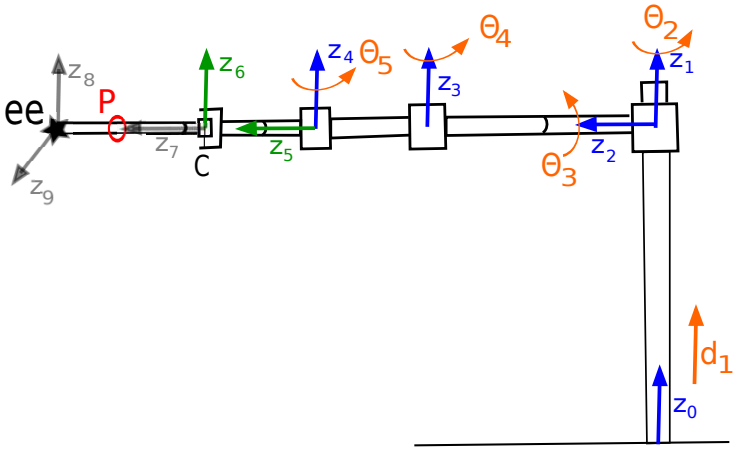

Figure 9: Laparoscopic robot. In orange, the different degrees of freedom of the robot, with their axes in blue. The passive joints' rotation axes are drawn in green, and the end effector's tool axes in grey. The trocar is marked in red.

link	a_i	α_i	d_i	θ_i	θ_i^{min}	θ_i^{max}
1	0	0	d_1+0.165	0	$-\pi/2$	$\pi/2$
2	0	$\pi/2$	0	$\theta_2' - \pi/2$	$-\pi/2$	$\pi/2$
3	a	$-\pi/2$	l_2	θ_3'	$-\pi/2$	$\pi/2$
4	0	0	0	θ_4'	$-\pi/2$	$\pi/2$
5	0	$\pi/2$	0	θ_5'	$-\pi/2$	$\pi/2$
6	0	$-\pi/2$	l_3	$\theta_6' + \pi/2$	$-\pi/2$	$\pi/2$
7	0	$\pi/2$	0	θ_7'	$-\pi/2$	$\pi/2$
8	0	$\pi/2$	l_4	$\theta_8' - \pi/2$	$-\pi/2$	$\pi/2$
9	0	$\pi/2$	0	$\theta_9' - \pi/2$	$-\pi/2$	$\pi/2$
10	0	$\pi/2$	0	$\theta_{10}' - \pi/2$	$-\pi/2$	$\pi/2$

where d_1, θ_2, θ_3, θ_4 and θ_5 are the *positioning* active degrees of freedom, θ_6 and θ_7 are two passive joints that ensure the robot enters the patient through the trocar (rotational around axes z_6 and z_7 in Figure 9) and θ_8, θ_9, θ_{10} are the joints forming a spherical wrist to orientate the end effector.

- Inverse kinematics equations
  To perform the inverse kinematics of the manipulator on the position (without considerating orientation), given a desired *end effector position*, X_{ee}, and assuming the trocar position is known (previously calculated), we have that in frame 0: $C^0 = X_{ee} + l_4 \frac{P - X_{ee}}{\|P - X_{ee}\|}$, and so, by the *Denavit-Hartenberg* parameters, we have

$$C^0 = T_1^0(d_1) \cdot T_2^1(\theta_2) \cdot T_3^2(\theta_3) \cdot T_4^3(\theta_4) \cdot T_5^4(\theta_5) \cdot \begin{pmatrix} 0 \\ 0 \\ l_3 \\ 1 \end{pmatrix}$$, which leads to the following equations:

$$\begin{aligned} C_x^0 &= c_2(l_3 c_3 c_4 s_5 + l_3 c_3 s_4 c_5 + a_1 c_3 c_4) + s_2(l_2 - a_1 s_4 - l_3(s_4 s_5 c_4 c_5)) \\ C_y^0 &= -c_2(l_2 - a_1 s_4 - l_3(s_4 s_5 c_4 c_5)) + s_2(l_3 c_3 c_4 s_5 + l_3 c_3 s_4 c_5 + a_1 c_3 c_4) \\ C_z^0 &= d_1 + (s_3 c_4 s_5 + s_3 s_4 c_5) l_3 + s_3 a_1 c_4 \end{aligned} \tag{9}$$

which can be combined to obtain:

$$(C_x^0 c_2 + C_y^0 s_2) s_3 + (z - d_1) c_3 = 0 \tag{10}$$

Now, having 5 degrees of freedom and a positioning task of 3 degrees of freedom, we can fix the values of d_1 and θ_2, to obtain two solutions for θ_3 in (10). These solutions can then be inserted onto (9) to obtain the other two angles θ_4 and θ_5. Again, we can see two assembly modes, one for each of the solutions of θ_3, both reaching the same position of C.

What we can see now is that we have to choose two of the degrees of freedom, thus having 2 dimentions to optimize the final position. What we can do is to calculate the passive joints with an initial solution and try to optimize, for example θ_5 when the other angles are known.

- Passive joints calculation.
  To obtain the passive joints θ_6 and θ_7, knowing joints $\theta_1, ..., \theta_5$ and the trocar point P, we can calculate the passive joints' values by calculating the end effector position in frame 5:

$$X_{ee}^5 = T_5^4(\theta_5)T_6^5(\theta_6)T_7^6(\theta_7)X_{ee}^7 = T_5^4(\theta_5)T_6^5(\theta_6)T_7^6(\theta_7)\begin{pmatrix} 0 \\ 0 \\ l_4 \\ 1 \end{pmatrix}$$

  and, on the other hand:

$$X_{ee}^5 = \left(T_5^0(\theta_1, ..., \theta_5)\right)^{-1} \cdot X_{ee}^0 =: \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix}$$

  Combining both expressions we have:

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} l_4 c_6 c_7 \\ l_4 s_6 s_7 \\ l_4 c_7 + l_3 \\ 1 \end{pmatrix}$$

  which gives the solution, if $\theta_7 \neq 0, \pi$:

$$\theta_6 = atan2(Q_y, Q_x)$$
$$c_7 = Q_z - l_3$$
$$s_7 = \begin{cases} Q_x/(l_4 c_6) & \text{if } \theta_6 \neq \pm\pi/2 \\ Q_y/(l_4 s_6) & \text{else} \end{cases}$$

- Analytical optimization.
  Nevertheless, in this kind of robots the positioning is recommended to be done so as to respect some restrictions, like the robot entering the patient from above, or to have a passive joint θ_7 as close to $\pi/2$ to optimize maneuverability. So assuming that d_1 only acts as an offset for the vertical height z, we can compute, given θ_2, θ_3, θ_4, the value of θ_5 that minimizes $\|\theta_7 - \pi/2\|$.

  To do so, knowing the trocar point in the base frame, P^0, and the first degrees of freedom, we have
  $P^4 = (T_4^0)^{-1}P^0 = \begin{pmatrix} x_4 \\ y_4 \\ z_4 \\ 1 \end{pmatrix}$ is known and, on the other hand:

Figure 10: Barrett's WAM arm

$$P_4 = T_5^4(\theta_5) \cdot T_6^5(\theta_6) \cdot T_7^6(\theta_7) \begin{pmatrix} 0 \\ 0 \\ \gamma \\ 1 \end{pmatrix} = \begin{pmatrix} \gamma c_5 c_6 s_7 + s_5(\gamma c_7 + l_3) \\ \gamma s_5 c_6 s_7 - c_5(\gamma c_7 + l_3) \\ \gamma s_6 s_7 \\ 1 \end{pmatrix}, \text{ where } \gamma = \|X_{ee} - P^0\|$$

Combining the first two terms we have: $x_4 s_5 - y_4 c_5 = \gamma c_7 + l_3$, which can be solved when $y \neq 0$ (details of this deduction have been considered not relevant) to get:

$$\theta_5 = atan2(x, -y) + acos\left(\frac{\gamma c_7 + l_3}{\sqrt{x^2 + y^2}}\right), \tag{11}$$

which has a solution when $|\frac{\gamma c_7 + l_3}{\sqrt{x^2+y^2}}| < 1$.

All these calculations might help when solving the inverse kinematics of this robot, but are not a closed form and the best way of calculating this inverse kinematics might be to use them in an iterative algorithm.

4.4.3 Barrett's WAM arm

The Barret's WAM arm (see Figure 10) is a 7-dof arm with the following *Denavit-Hartenberg* parameters:

link	a_i	α_i	d_i	θ_i	θ_i^{min}	θ_i^{max}
1	0	$-\pi/2$	0	θ_1	-2.6	2.6
2	0	$\pi/2$	0	θ_2	-2.0	2.0
3	a	$-\pi/2$	d_3	θ_3	-2.8	2.8
4	-a	$\pi/2$	0	θ_4	-0.9	3.1
5	0	$-\pi/2$	d_5	θ_5	-4.8	1.3
6	0	$\pi/2$	0	θ_6	-1.6	1.6
7	0	0	d_7	θ_7	-2.2	2.2

where $d_3 = 0.55$, $d_5 = 0.3$ and $d_7 = 0.06$

Now, for solving its inverse kinematics, we have to keep in mind that this robot has one redundant degree of freedom. So what can be done is to solve the system with one fixed angle, and then perform some optimization. The value of this fixed angle may come for a reference

position, such as could be a *rest* position, or just the previous position in a trajectory to be performed.

It is also relevant in this robot that the last 3 dof consist of a *Spherical wrist*, which can then be treated independently. So to compute the inverse kinematics of the robot given a desired homogeneous transformation:

$$T_{obj} = \begin{pmatrix} R_x & R_y & R_z & P_{obj} \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{12}$$

we can consider the spherical wrist center as $P_w = P_{obj} - d_7 \cdot R_z$.

This reduces the 7-variable problem to a 4-dof positioning problem, where the task is to reach point P_w (dimension 3), without paying attention (*a priori*) to the orientation of the end effector.

Now, back to the problem, we have a 4 dof robot with a 3-D task. And the first question to ask is which angle to fix to calculate a first solution.

When looking at the robot, one might think that the *closest* angle to represent the redundancy of this robot might be θ_3. But in fact, after comparing the solutions of fixing any of the first 3 joints, it has been concluded that the best joint to be fixed when solving the inverse kinematics is the first one, mainly because it moves all the inertia of the robot, and minimizing its angle variation will reduce the total energy in a movement.

First, we can compute the angle θ_4 with the equation of the distance from the origin to the spherical wrist point:

$$\begin{aligned} \|P_w^0\|^2 = \|T_1^0(\theta_1) \cdot T_2^1(\theta_2) \cdot T_3^2(\theta_3) \cdot T_4^3(\theta_4)\|^2 = ... \\ = d_3^2 + d_5^2 + d_7^2 + 2(d_5 \cdot a + a \cdot d_3)sin(\theta_4) + 2(d_3 \cdot d_5 + a^2)cos(\theta_4), \end{aligned}$$

which can be solved to obtain two possible values of θ_4, called *inner elbow* and *outer elbow* configurations, depending on the sign of θ_4.

And then, knowing that $P_w^4 = \begin{pmatrix} 0 \\ 0 \\ 0.3 \\ 1 \end{pmatrix}$, we can express $T_3^2(\theta_3)T_3^4(\theta_4) \cdot P_w^4 = (T_2^1(\theta_2))^{-1}(T_0^1(\theta_1))^{-1} \cdot P_w^0$:

$$\begin{bmatrix} 0.3\,c_3 s_4 - \frac{9}{200}\,c_3 c_4 + \frac{9}{200}\,c_3 \\ 0.3\,s_3 s_4 - \frac{9}{200}\,s_3 c_4 + \frac{9}{200}\,s_3 \\ \frac{11}{20} + 0.3\,c_4 + \frac{9}{200}\,s_4 \\ 1.0 \end{bmatrix} =$$

$$= \begin{bmatrix} s_1 c_2 (P_w^0)_x + s_1 c2 (P_w^0)_y - s_2 (P_w^0)_z \\ -s_1 (P_w^0)_x + c_1 (P_w^0)_y \\ c_1 s_2 (P_w^0)_x + s_1 s_2 (P_w^0)_y + c_2 (P_w^0)_z \\ 1 \end{bmatrix},$$

from where, knowing θ_1 and θ_4, we can obtain the two solutions of θ_2 by solving:

$$\frac{11}{20} + 0.3\,c_4 + \frac{9}{200}\,s_4 \;=\; c_1 s_2 (P_w^0)_x + s_1 s_2 (P_w^0)_y + c_2 (P_w^0)_z \tag{13}$$

and, with θ_2, we obtain $c_3 = cos(\theta_3)$:

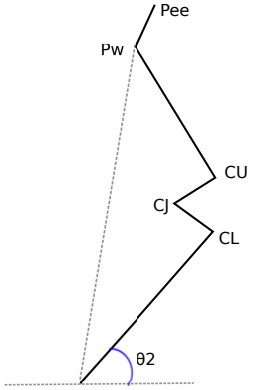

Figure 11: Barrett's WAM arm

$$0.3\, c_3 s_4 - \frac{9}{200}\, c_3 c_4 + \frac{9}{200}\, c_3 = s_1 c_2 (P_w^0)_x + s_1 c2 (P_w^0)_y - s_2 (P_w^0)_z \tag{14}$$

and $s3 = sin(\theta_3)$:

$$0.3\, s_3 s_4 - \frac{9}{200}\, s_3 c_4 + \frac{9}{200}\, s_3 = -s_1 (P_w^0)_x + c_1 (P_w^0)_y \tag{15}$$

To finally get $\theta_3 = atan2(sin(\theta_3)$**,**$cos(\theta_3))$. The rest of the angles (θ_5,θ_6,θ_7) are obtained as a spherical wrist.

This set of equations finds, if possile, the joint position of the WAM arm verifying $\theta_1 = \theta_1^0$. If no solution is found, another value of θ_1^0 can be used to find a solution. Usually starting at a reference value and making bigger changes on it until a solution is found.

In [11] one can find a package with an inverse kinematic algorithm that performs this search around θ_3. This algorithm searches for the solution with the closest θ_3 angle to the initial one. As we have commented, we find it more adequate to search over θ_1.

Now, to perform an optimization of the joints position of the robot, knowing a first solution of its kinematics, [18] and [21] propose that, for these kind of robots their redundant degree of freedom comes from the elbow rotating about the axis s from the base of the robot to its wrist.

To do such a rotation, knowing points CL, CU, CJ as defined in the Figure 11, we can rotate them an angle ϕ around s with the equation:

$$C_\phi = R_\phi C$$

being $R_\phi = I_3 + sin(\phi)\,[u_{sw}\times] + (1 - cos(\phi))\,[u_{sw}\times]^2$, being u_{sw} the skew-symetric matrix of a vector.

Note that u_{sw} is an eigenvector of eigenvalue 1 of R_ϕ:

$$R_\phi \cdot u_{sw} = u_{sw} + sin(\phi)\,[u_{sw}\times]\,u_{sw} + (1 - cos(\phi))\,[u_{sw}\times]^2\,u_{sw} = u_{sw}, \text{ as } [u_{sw}\times]\,u_{sw} = 0$$

And so, we obtain the new rotated points CL_ϕ, CU_ϕ and CJ_ϕ.

We can compute the new angles $\theta_1 = atan2(CL_y, CL_x)$ and $\theta_2 = acos(CL_z/d_3)$, and θ_3 solving the system:

$$(CJ - CL)/a = \begin{bmatrix} cos(\theta_1)cos(\theta_2) & -sin(\theta_1) \\ sin(\theta_1)cos(\theta_2) & cos(\theta_1) \\ -sin(\theta_2) & 0 \end{bmatrix} \cdot \begin{bmatrix} cos(\theta_3) \\ sin(\theta_3) \end{bmatrix}$$

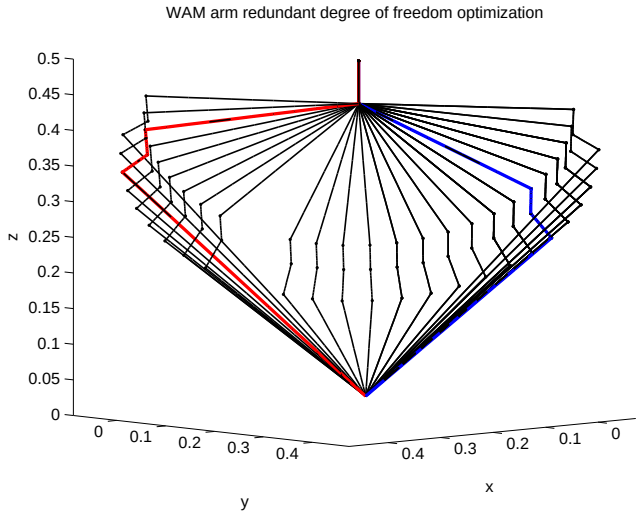

Figure 12: WAM arm optimization. In red, the first solution found. In blue, the optimal solution, and in black, all the solutions found and evaluated.

As the end effector's position has not changed with a rotation of the solution, the angle θ_4 will be the same if the elbow configuration (sign of θ_4) is constant.

Note that, if $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]$ is a solution of the inverse kinematics, so it is $[\theta_1 \pm \pi, -\theta_2, \theta_3 \pm \pi, \theta_4, \theta_5 \pm \pi, -\theta_6, \theta_7 \pm \pi]$; thus for each solution on θ_4, we have 4 solutions of the inverse kinematics, where it is recommended to use the one closer to a reference position (or the previous position), and then one can solve for the rest of the angles as a spherical wrist, to obtain new rotated solutions, and so perform an optimization of a defined cost function depending on ϕ.

To do such a performance, we have decided to take a weighted norm of the solution vector $\|\theta_{sol} - \theta_{init}\|^2 = (\theta_{sol} - \theta_{init})^T \cdot diag(2.5, 2, 2, 1.5, 1, 1, 1) \cdot (\theta_{sol} - \theta_{init})$, with θ_{init} a vector of zeros and perform optimization to minimize this norm. In fact, with the equations exposed in this section we find a first solution being the one minimizing this norm, but with a weighting matrix of $W = diag(1, 0, 0, 0, 0, 0, 0)$, and the one on the commented [11] is the one satisfying $W = diag(0, 0, 1, 0, 0, 0, 0)$. Nevertheless, any optimization function can be used.

In Figure 12 we see an example of this rotations, using the joint vector norm as an optimization criteria.

5 Time-discrete control-based methods

Sometimes it is not possible to find a closed-form analytical solution to the inverse kinematic problem of a manipulator, and this becomes even harder in the case of a redundant manipulator, where infinite solutions may exist, forming a subset of the joint space. In these cases, control methods are often applied to solve the problem of the inverse kinematics of the manipulator.

In this section, we analyse the state of the art on inverse kinematics with control methods, commenting the advantages and disadvantages of the different methods on the literature, and we try to find a method which is robust when handling joint limits, convergence issues and singularities.

We must point out that all these methods are first-order derivative methods, whose systems are linear. There are also other linear methods such as the extended Jacobian (proposed initially by Baillieul [1] and further developed in [9]), which are not considered here because of their need to compute the 2nd order Jacobian, which makes the computational complexity of the method grow.

5.1 General schemes

The typical inverse kinematics method has a scheme as in Figure 13, where a desired position x_d is taken as a reference, and comparing to the actual state x, an error e is obtained. This error is then used in a function h, not necessarly lineal, to obtain a joint state θ which is then used to, with the forward kinematics (f in the figure) of the manipulator, obtain the new state x.

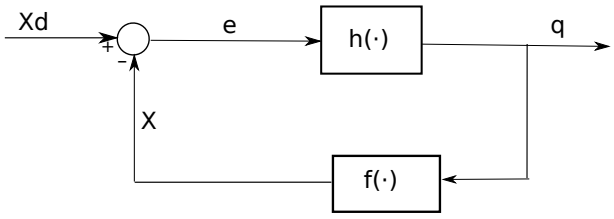

Figure 13: General scheme of an inverse kinematics controller

5.2 About the convergence of control-based methods

A major problem found in the literature is that, in many cases, convergence of a method is demonstrated by means of its continuous version, taking a valid Lyapunov function which indeed, demonstrates its stability. But in a discrete-time controller, as in these results might not be valid. In fact, as the solution to the inverse kinematics problem is calculated as $\theta^{k+1} = \theta^k + \Delta\theta$, for some computed $\Delta\theta^k = \alpha h(\theta^k)(x_d - f(\theta^k))$ and a step α which is usually a computation step T times a scalar step γ. A continuous-time solution would mean an infinitely small step α which, by computation restrictions, is impossible.

Although there exist discrete-time versions of Lyapunov Theory, such as [8], its application is not as trivial as it would be in a continuous-time problem.

There is also some literature about the convergence of these methods which takes the discrete-time system as a succesion and proves its convergence. [7] finds an upper bound of the step α which guarantees convergence, but restricting the operational space to a subset where the Jacobian is full-rank, smooth and bounded, so its application is not general. Nevertheless, it points out the relevance of the initial error dependency of these methods to converge, so the closer to the goal, the better these methods perform, as we will see later.

The problem of finding a step which, on the one hand, ensures convergence and, on the other hand, gives an acceptable computational cost, is then an issue to focus on.

Using first order derivative methods of the robot's motion has the drawback that, depending on the method and the goal position, an algorithm can get stuck at an *algorithmic singularity*, which is when we arrive at a point where the error e belongs to the kernel of the inverted Jacobian, or in a multiple-task method, a secondary task joint variation takes the contrary value of the primary task.

5.3 Methods from literature

In the next sections, we will describe different control-based methodologies for obtaining the inverse kinematics, we will apply them to a simple robot such as a 4R with all joint limits set at $[-\pi/2, \pi/2]$. Most of the methods are based on the pseudoinverse of the Jacobian. With this, one may ask why not using J^{-1}. In fact, the inverse matrix does not exist for a non-squared or non-full ranked matrices and also its terms tend to infinite as the matrix approaches a singularity. To our criteria, we will mainly prioritize the following:

- Desired position convergence.
- Joint limit avoidance.
- Robustness when a singularity occurs.
- Computational cost.

From now on, a set of methods will be described. For each of them, we will first explain its main idea intuitively, and then present its algorithm, to later on show their behaviour with examples and comment their advantages and disadvantages. To illustrate, we will use a 4R planar robot for a positioning 2-dimensional task, which gives a remaining of 2 redundant degrees of freedom with the following *Denavit-Hartenberg* parameters:

link	$a_i(m)$	α_i	d_i	θ_i	θ_i^{min}	θ_i^{max}
1	0.4	0	0	θ_1	0	0
2	0.3	0	0	θ_2	0	0
3	0.2	0	0	θ_3	0	0
4	0.1	0	0	θ_4	0	0

To start, we will present the two most simple methods: the Jacobian pseudoinverse and the Jacobian transposed. We will comment their drawbacks, and we will present some variations over those methods so as to reduce the effect of their main concerns, which are:

- **Jacobian with very small eigenvalues (singularities)**. To solve this problem, the typical solution applied is to damp or filter the Jacobian matrix when calculating its inverse, or even damp selectively according to the joint's movements.
- **Secondary tasks**. When possible, secondary tasks, apart from reaching the goal can be added to an algorithm. To this purpose, we present the gradient projection method, its generalisation, the task-priority methods, and also the Jacobian weighting methods.
- **Joint limit avoidance**. To avoid joint limits, apart from considering it as a secondary task or even a primary one, joints can be blocked when necessary. This can lead to a discontinuous activation and deactivation of blocking tasks, so a continuous clamping is also presented.

5.3.1 Jacobian pseudoinverse

The Jacobian pseudoinverse method consists in directly inverting the matrix J in the first-order differential equation $\dot{x} = J\dot{\theta}$ (discretized as $\Delta x = e = J\Delta\theta$. The point is that, on redundant manipulators, J is not a square matrix, so a more general inverse matrix, called Moore-Penrose pseudoinverse, is used so as to invert J and solve the problem. This inverse operator has also the property to give the least-squares minimum norm solution. so the solution $\Delta\theta$ gives the minimum value of $\|J\Delta\theta - e\|$.

As we have just said, the Jacobian pseudoinverse method is based on the Moore-Penrose pseudoinverse matrix, which is defined as $J^\dagger = J^T \cdot (J \cdot J^T)^{-1}$ when $J \cdot J^T$ is invertible. For non full row rank matrices, the pseudoinverse is defined more generally as the matrix verifying $JJ^\dagger J = J$.

With this matrix, we define the method as:

$$\Delta\theta = \alpha \cdot J^\dagger \cdot e,$$

being α the method step, $J^\dagger$ the pseudoinverse of the geometric Jacobian of the manipulator, and e the task error.

In some cases, when a robot is in a singularity, the method might get stuck in a point or have too large values on the pseudoinverse matrix. This can be easily seen if we take the formula of the Jacobian pseudoinverse matrix: $J^\dagger = J^T \cdot (J \cdot J^T)^{-1}$. If the Jacobian matrix is close to a singularity, then $J \cdot J^T$ will have very small terms, which will translate to very large terms when inverting the matrix.

To verify that, if we take the singular value decomposition of J:

$$J = U \cdot \Sigma \cdot V^T, \tag{16}$$

where U, V are orthonormal matrices (if J is $n \times m$, being m the number of joints and n the task space dimension, then U is $m \times m$ and V is $n \times n$) and Σ is a diagonal $m \times n$ matrix with σ_i its diagonal elements, where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_m \geq 0$. (Note that σ_i may be zero if J is rank-deficient).

Then if we define $r = max_i\{i \mid \sigma_i > 0\}$, we have that $r = rank(J)$, and we can write (considering $m \geq n$):

$J = \sum_{i=1}^{m} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$, with $\mathbf{u}_i$ and $\mathbf{v}_i$ being the ith columns of U and V, respectively.

Now, as U and V are orthonormal matrices, we can compute the pseudoinverse of J as

$J^\dagger = V \cdot \Sigma^\dagger \cdot U^T = \sum_{i=1}^{r} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T$.

With this expression of the Jacobian pseudoinverse, we clearly see that when the robot gets close to a singularity, one σ_j becomes very small and so $\frac{1}{\sigma_j}$ can be very large. As $\lim_{\sigma_j \to 0} \frac{1}{\sigma_j} = \infty$, this implies very large gains when computing $\Delta\theta = J^\dagger \cdot e$.

With this representation, the *reconstruction error* of the method, i.e. the difference between the error and the variation of position in the task space, at a differential level, can be expressed, for $\alpha = 1$ as [4]:

$\dot{x}_e - J \cdot \dot{q} = \sum_{i=r+1}^{m} \frac{\mathbf{u}_i^T \cdot \dot{\mathbf{x}}_e}{\sigma_i} \mathbf{v}_i$.

Now we will show the results of the pseudoinverse method, using the scheme $\Delta\theta = J^\dagger e$, with a step $\alpha = 1$.

The three results shown are those obtained in a planar 4R robot, with a position tracking

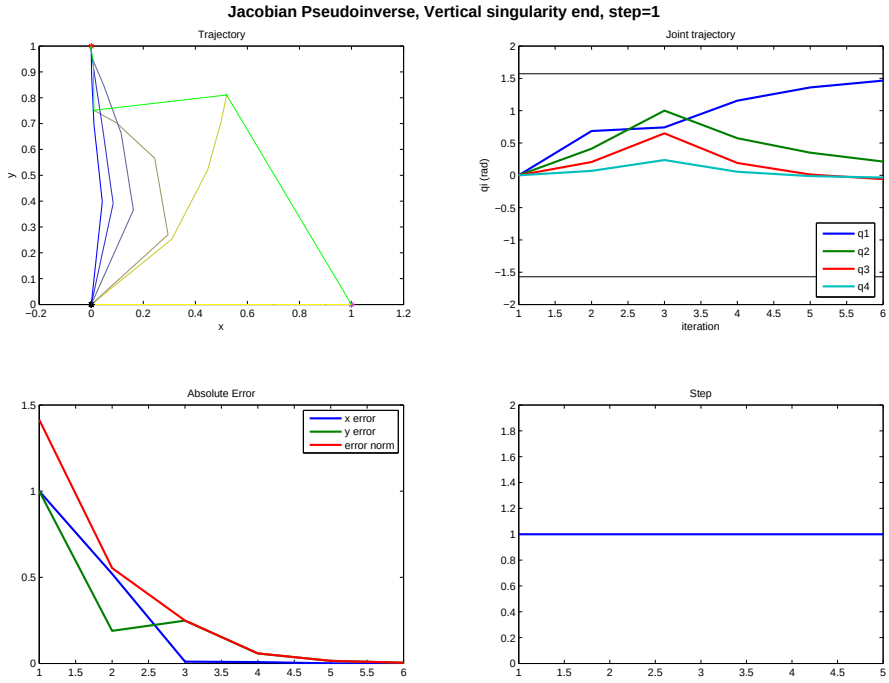

Figure 14: Behaviour of the pseudoinverse method in a singular objective. On the upper left plot, the green line shows the evolution of end effector's position, while the robot structure is drawn evolving from yellow to blue.

(not considering orientation) for an initial configuration of $\theta = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ and three different desired configurations, with a tolerated error of 1 cm. This error might be considered too high for real tasks, but we have considered it is small enough to these illustration purposes.

The results are shown in 4 plots, as we can see in Figure 14: The first of them is the trajectory of the robot, plotted in yellow for the initial position, and evolving to blue, whith its end effector's trajectory plotted in green. The upper left plot corresponds to the individual trajectory of each joint, with their limits marked as horizontal lines. The lower-left plot corresponds to the error of the end-effector positioning, and last, the lower-right plot shows the step α taken at each iteration.

- Vertical singular end position.

  In this case (Figure 14), we show the results for a desired goal $x_d = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The algorithm is very fast, with a computation time of 0.0630s. Nevertheless, as we can see in the figure, the first step is a big leap.
- Joint limit forcing.

  Now the desired goal is $x_d = \begin{pmatrix} 0 \\ 0.1 \end{pmatrix}$, what causes the robot to quickly approach its joint limits, set at $[-\pi/2, \pi/2]$ for all joints. As it could be expected, now the joint limits are not respected, as seen in Figure 15.
- Horizontal end position. In this case, what happens in Figure 16 is that the robot does

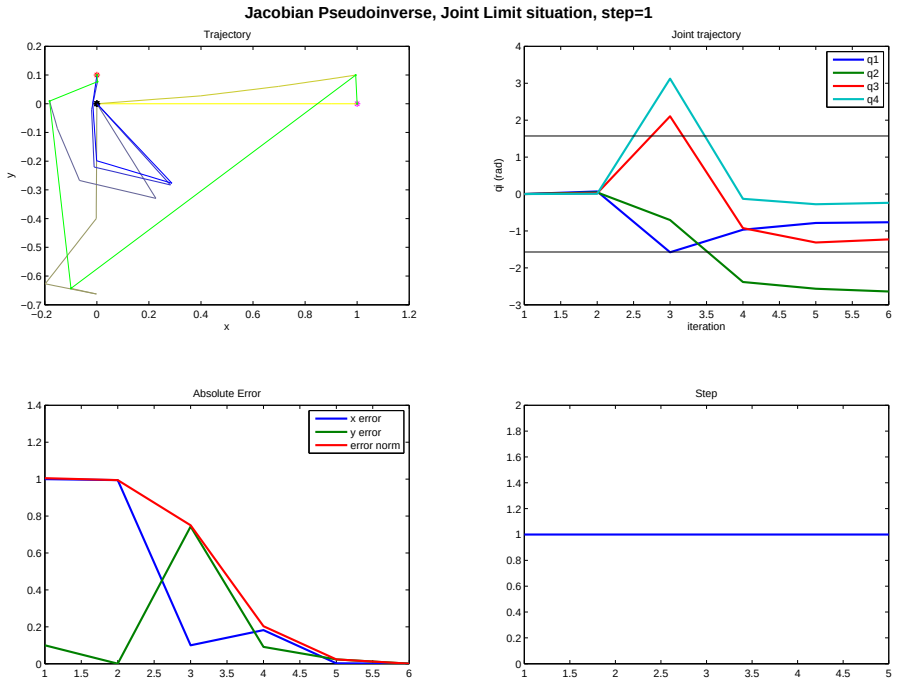

Figure 15: Behaviour of the pseudoinverse method when the goal vulnerates joint limits

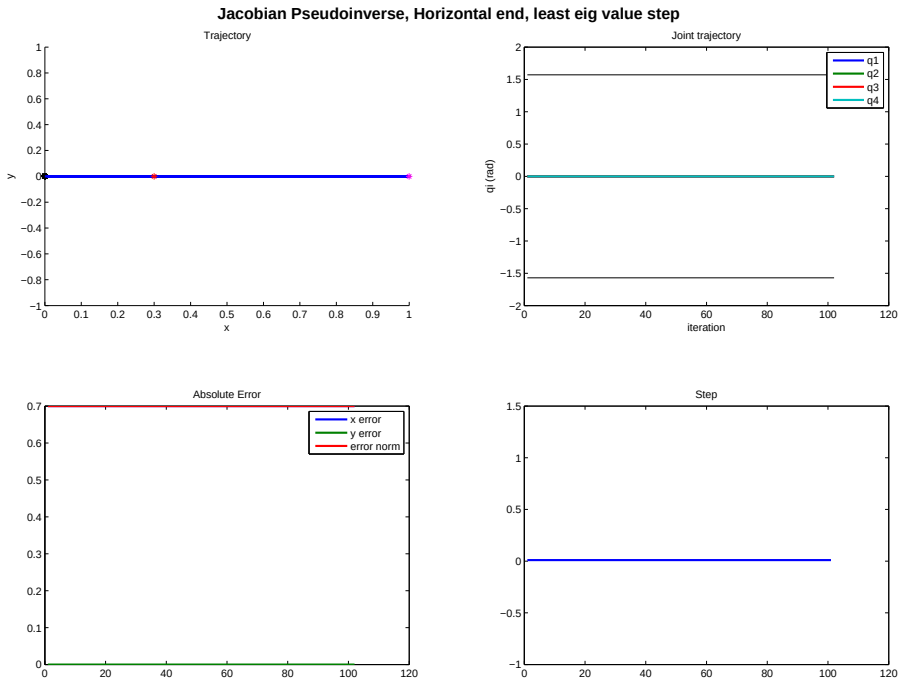

Figure 16: Behaviour of the Jacobian pseudoinverse method on a controllability singularity

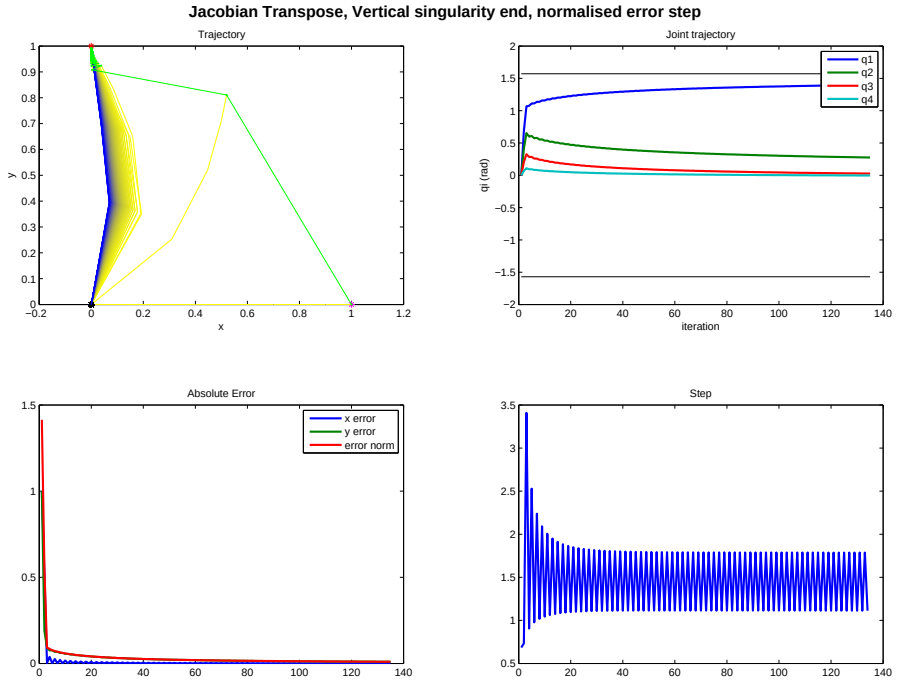

Figure 17: Behaviour of the Jacobian transpose method when the goal is a singular position

not move at all. The Jacobian matrix in this case has the form $J = [\ \mathbf{0} \quad \mathbf{v}\ ]$ and the error vector $e = \ [e_x \quad 0]^T$. When we calculate the pseudoinverse of J, it only has rank 1 so its product with e is 0 and the robot won't move.

5.3.2 Jacobian transpose

To gain speed, the Jacobian transpose method uses, instead of an inverse of the matrix J, its transpose. This might be compared as if we were considering matrix J as a kind of orthonormal matrix. Nevertheless, it can be proved in terms of Lypaunov theorem that, for a sufficiently small step α, the control scheme converges to zero error.

As said, now the control rule using the scheme in Figure 13 is: $\Delta\theta = \alpha J^T e$,where J^T is now the transpose of the geometric Jacobian of the manipulator.

This method is computationally very fast step by step, although it may require more steps than other methods, and it does not consider joint limits.

When choosing α, some literature recommend taking the value that would make the norm of the change in the task space equal to the norm of the error. That is $\alpha = \frac{<JJ^Te,e>}{<JJ^Te,JJ^Te>}$.

Now we will show some examples of the behaviour of this method for the mentioned 4R manipulator, using (with $< \cdot, \cdot >$ the dot product):

$$\Delta\theta = \frac{< JJ^Te, e >}{< JJ^Te, JJ^Te >} J^Te. \tag{17}$$

- Vertical singular end position.
  In Figure 17, the computation time has been 0.3085s, which is quite fast, althought it presents some *chattering* when reaching the goal. This is due to the fact that the second

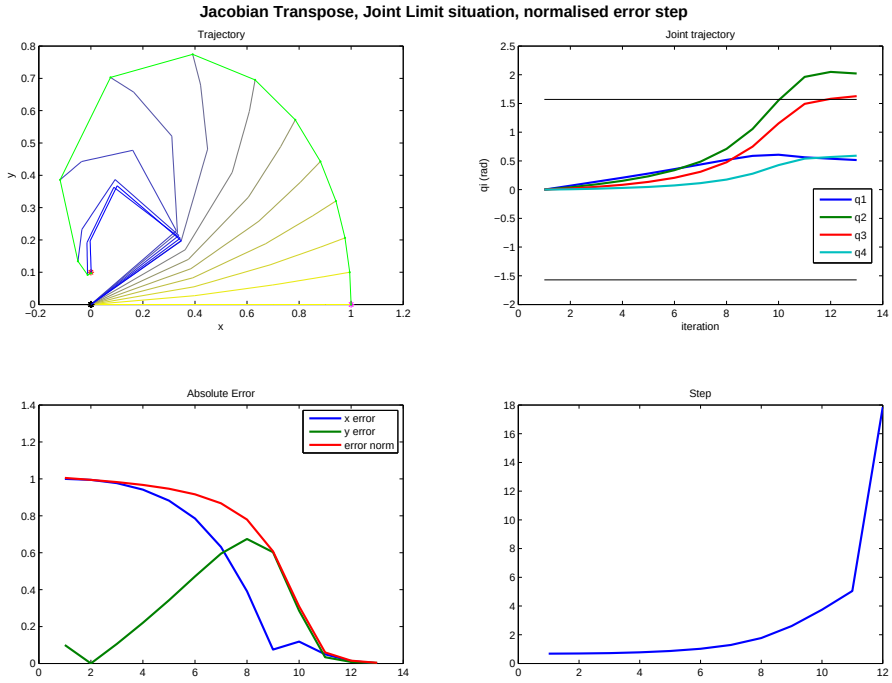

Figure 18: Behaviour of the Jacobian transpose method when the objective vulnerates joint limits

row of the Jacobian matrix has very small values, compared to the first row, as the robot approaches the singularity on the goal, so a gain on vertical position implies more variation in the horizontal direction. This chattering makes the algorithm find the solution much slower, loosing the computational speed gained in avoiding to invert the Jacobian matrix.

- Joint limit forcing.
  In Figure 18, we see that, as the Jacobian transposed method does not take into account joint limits, those are easily surpassed.
  What could be done to avoid surpassing these joint limits is to assign the maximum or minimum value for the joint if it is surpassed, but this does not work well, as the joint is forced to keep moving towards the joint limit and it gets blocked, possibly blocking the other joints, as we can see in Figure 19.
- Horizontal end position.
  Now the same problem as with the Jacobian pseudoinverse method happens when the horizontal position is the goal, avoiding the robot to move in that direction.

5.3.3 Jacobian damping and filtering

As commented on the Jacobian pseudoinverse method, when close to a singularity, the pseudoinverse method can have very large gains. In addition, the pseudoinverse operator has a discontinuity w.r.t. the eigenvalues of J in $\sigma = 0$. To avoid this discontinuity and reduce these large gains, which imply a large condition number of the Jacobian, meaning numerical error, the Jacobian pseudoinverse can be damped.

There are some ways of trying to avoid discontinuities on the singularities, such as damping/filtering the Jacobian matrix, a good view of these methods can be found in [6] and [4].

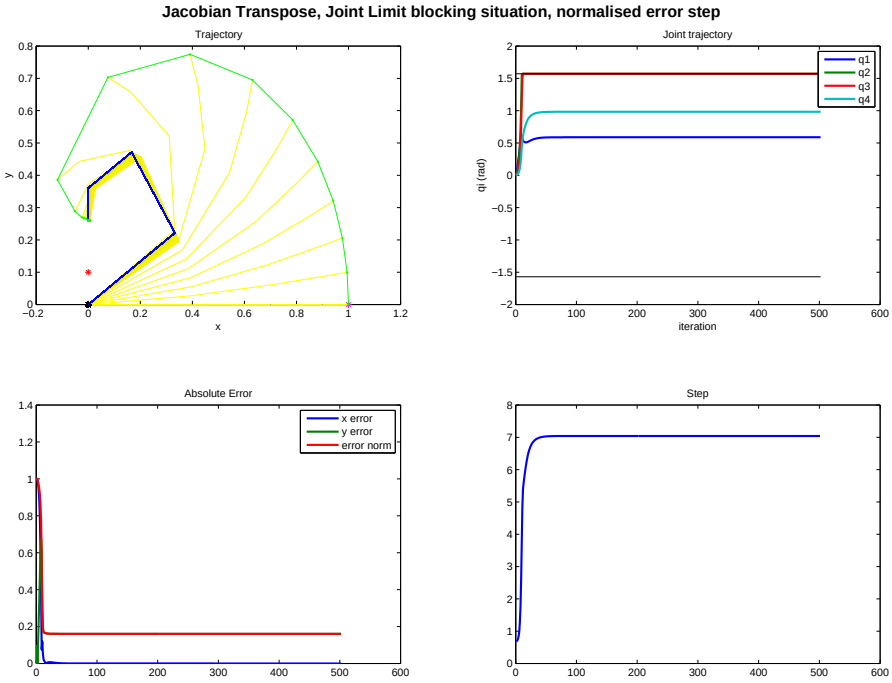

Figure 19: Behaviour of the Jacobian transpose method with joints being blocked at their limits

- **Jacobian damping**. If we take an alternative expression of the pseudoinverse, called *damped pseudoinverse*, defined as: $J^{\hat{\dagger}} = J \cdot (JJ^T + \lambda^2 I)^{-1} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2+\lambda^2} \mathbf{v}_i \mathbf{u}_i^T$
  which is, for some small λ, almost the same matrix as the ordinary pseudoinverse when $\sigma_i^2 \gg \lambda^2 \ \forall i$, and when some σ_j is close to zero, $\lim_{\sigma_j \to 0} \frac{\sigma_i}{\sigma_i^2+\lambda^2} = 0$, instead of ∞.
  This avoids the large gains commented, but also adds a *chattering* to the solution when close to a singularity. This chattering must be avoided, commonly picking a very small value of λ, as the new reconstruction error will be, for $\alpha = 1$
  $\dot{x}_e - J \cdot \dot{q} = \sum_{i=1}^{r} \lambda^2 \frac{(\mathbf{u}_i^T \dot{\mathbf{x}}_e) - \sigma_i (\mathbf{v}_i^T \dot{\mathbf{q}}_0)}{\sigma_i^2+\lambda^2} \mathbf{v}_i + \sum_{i=r+1}^{m} \frac{\mathbf{u}_i^T \cdot \dot{\mathbf{x}}_e}{\sigma_i} \mathbf{v}_i$
  which evidentiates the possible need of varying the damping factor λ, as it adds a new error. This is also commented in [6].
  Now, testing this method on the 4R manipulator: $\Delta\theta = \alpha J \cdot (JJ^T + \lambda^2 I)^{-1} e = \alpha \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2+\lambda^2} \mathbf{v}_i \mathbf{u}_i^T e$
  - Vertical singular end position, as seen in Figure 20, with a computation time of 0.0525s. The method presents no problems as expected.
  - Horizontal end position. The same as in Figure 16 happens. This is due to the fact that the damping is effective in a neighbourhood of a singularity, but, looking at the singular value decomposition of the method, we can see that $\frac{\sigma_i}{\sigma_i^2+\lambda^2} = 0$ for $\sigma_i = 0$.

- **Numerical filtering**. A way of reducing the mentioned reconstruction error in the Jacobian damping methods is to define a singular region and apply the damping factor only when entering it [5].

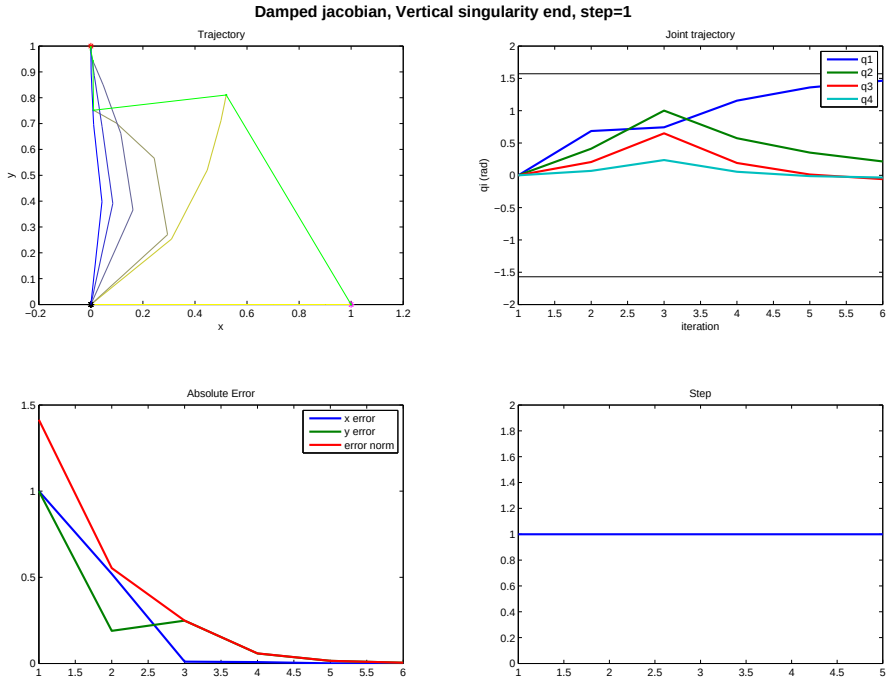

Figure 20: Damped Jacobian method in a vertical singularity end position. The method converges

To this purpose, we define a variable damping factor as

$$\lambda^2 = \begin{cases} 0 & \text{if } \sigma_n \geq \epsilon \\ (1 - (\frac{\sigma_n}{\epsilon})^2)\lambda_{max} & \text{if } \sigma_n < \epsilon \end{cases} \tag{18}$$

where σ_n is the smallest eigenvalue of the Jacobian matrix, ϵ is the width of the singular region (in terms of singular values) in which the damping factor takes non-zero value, and λ_{max} is the maximum damping factor. And so, the new filtered pseudoinverse matrix is $J^{\ddagger} = J \cdot (JJ^T + \lambda^2 \mathbf{u}_n \mathbf{u}_n^T)^{-1}$, where $\mathbf{u}_n$ is the eigenvector corresponding to the smallest eigenvalue.
Again, now the reconstruction error at a differential level, for a step of $\alpha = 1$ is
$\dot{x}_e - J \cdot \dot{q} = \lambda^2 \frac{\mathbf{u}_m \dot{\mathbf{x}}_e - \sigma_m (\mathbf{v}_m^T \dot{\mathbf{q}}_0)}{\sigma_m^2 + \lambda^2} \mathbf{u}_m$.
Testing it for the 4R planar robot: $\Delta\theta = \alpha \sum_{i=1}^{n-1} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T e + \frac{\sigma_n}{\sigma_n^2 + \lambda^2} \mathbf{v}_n \mathbf{u}_n^T e$

- Vertical singular end position. See Figure 21. As it could be expected, this case presents no difficulties.
- Horizontal end position. Figure 22 As in the damped least squares, the effect is the same as in Figure 16. But if we add a random perturbation of order 10^{-6}, this method allows the convergence to the goal, although it does so with very large gains.

5.3.4 Selectively damped least squares.

As we have seen, some methods make big leaps in their firs steps. This is due to the fact that a small eigenvalue causes the Jacobian pseudoinverse method to have big gains in all directions,

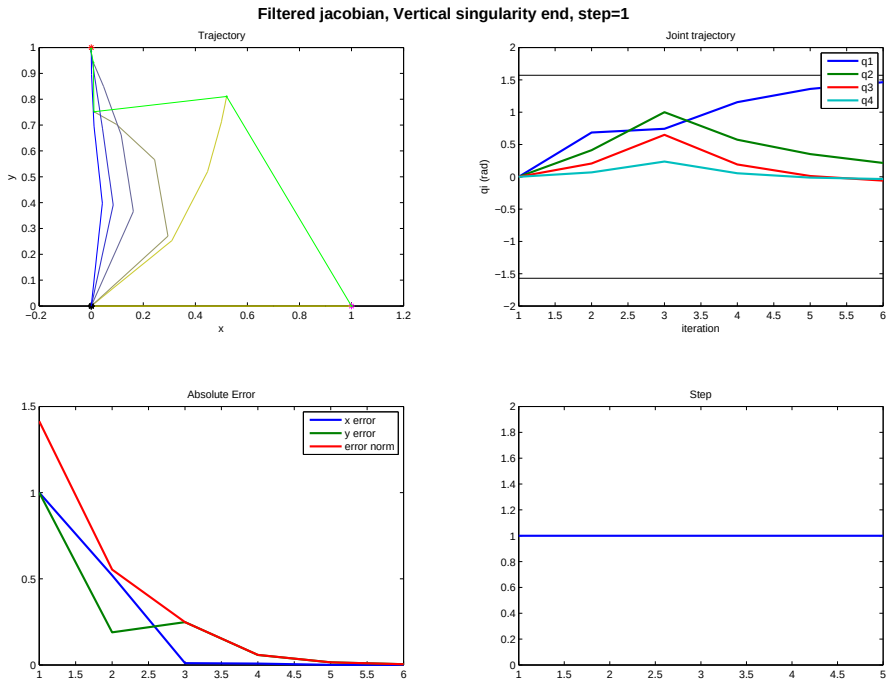

Figure 21: Filtered Jacobian method.

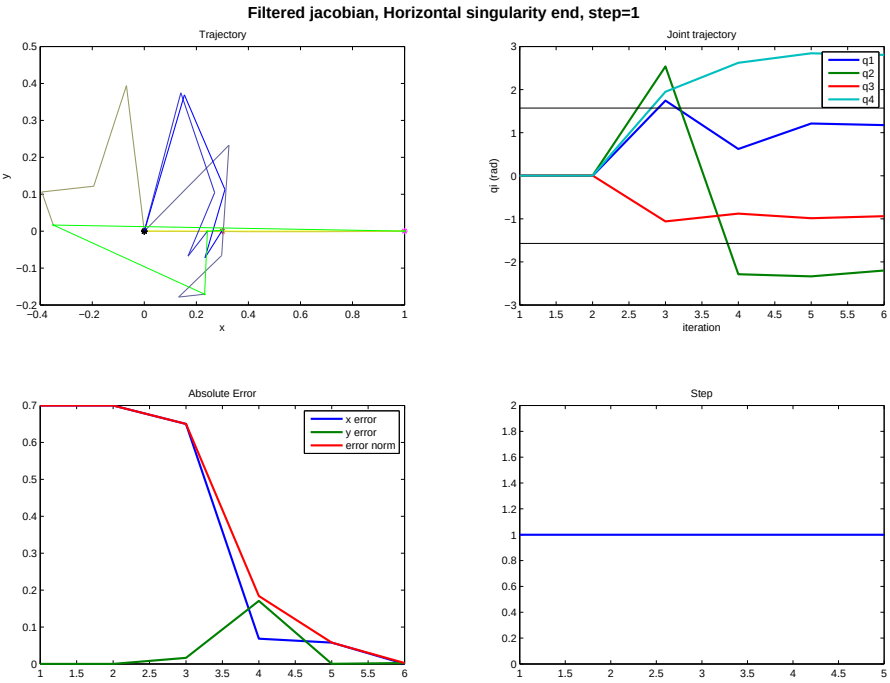

Figure 22: Filtered Jacobian method with a perturbation on the initial singular position.

even if these small eigenvalues are damped or filtered. In [2], a new way of damping is defined, which consists in damping differently the effect of each one of the components of the position error, expressed in the base of the singular value decomposition of J, so small eigenvalues of the Jacobian, which would turn into large gains, are damped more severely.

First, consider the singular value decomposition of the Jacobian matrix, and the pseudoinverse algorithm:

$$J = U\Sigma V^T = \sum_{j=1}^{n} u_i \sigma_i v_i^T.$$

$$\Delta\theta = J^{\dagger} e = \sum_{j=1}^{n} \sigma_i^{-1} v_i u_i^T e,$$

where u_i are the eigenvectors of J in the task space.

Then if we consider a unitary error in the direction of one of the eigenvectors in the task space (columns of U), $e = u_i$, then the pseudoinverse method would give a variation of joint j of $\Delta\theta_j = \sigma_i^{-1} v_{j,i}(u_i^T \cdot u_i) = \sigma_i^{-1} v_{j,i}$, which translates in a movement of the robot in the direction of u_i of $\sigma_i^{-1} v_{j,i} J_j$, being J_j the jth column of the Jacobian matrix. Thus it is defined:

$$M_i = \sigma_i^{-1} \sum_{j=1}^{m} |v_{j,i}| \|J_j\|, \tag{19}$$

where J_j is the jth column of the Jacobian matrix of the manipulator, and M_i estimates the sum of the distances moved by the end effector caused by the individual changes in joint angles.

Now, defining a maximum angle change γ_{max}, we can bound the angle change in response to the ith component (in base U) of the error e as:

$\gamma_i = min(1, 1/M_i)\gamma_{max}$.

The idea is that, when M_i is small, the joints' changes of each error component have cancelled each other, and so a higher response is needed, but only in the affected direction (u_i), as with the pseudoinverse method the same gain is applied to all coordinates.

Now this γ_i is used to damp the gain on each column of matrix U

$$\Delta q_i = \begin{cases} w_i & \text{if } \|w_i\|_\infty \leq \gamma_i \\ \gamma_i \frac{w_i}{\|w_i\|_\infty} & \text{otherwise} \end{cases}$$

being $w_i = \sigma_i^{-1} v_i (u_i^T \cdot e)$ ($w_i = 0$ in case $\sigma_i = 0$)

And finally damping the total motion by

$$\Delta\theta = \begin{cases} \Delta\hat{\theta} & \text{if } \|\Delta\hat{\theta}\|_\infty \leq \gamma_{max} \\ \gamma_{max} \frac{\Delta\hat{\theta}}{\|\Delta\hat{\theta}\|_\infty} & \text{otherwise} \end{cases}$$

Where $\Delta\hat{\theta} = \sum_{i|\sigma_i \neq 0} \Delta q_i$

Now, using this method with a 4R planar robot, we have:

- Vertical singular end position. Computation time of 0.0642s. As we can see in Figure 23, the method converges fastly, and smoothly.
- Joint limit forcing. (Computation time of 0.0890s) Although the steps are very uniform, as we can see in Figure 24, joint limits are not respected because they are not considered in the algorithm.
- Horizontal end position. In this case, it also happens to be that the initial error belongs to the kernel of the algorithm, so the robot won't move as in Figure 16.

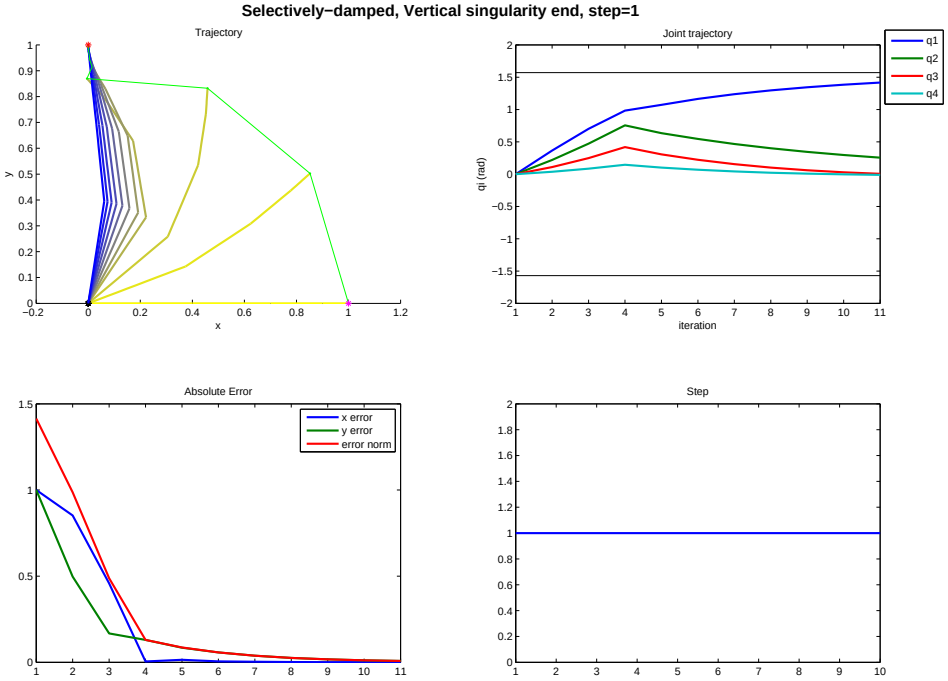

Figure 23: selectively damped least squares method. Vertical singular goal.

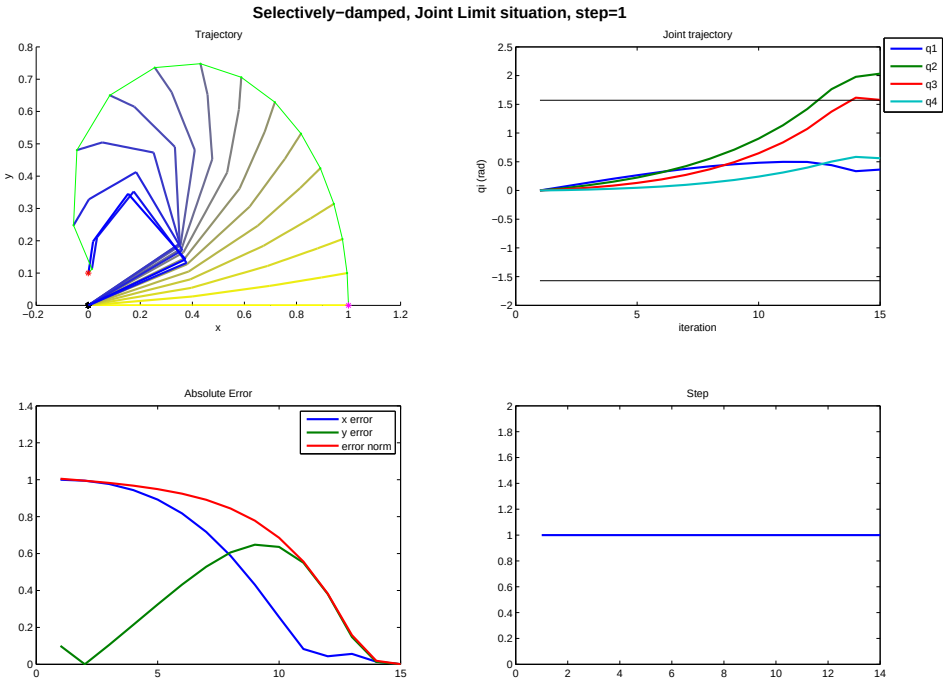

Figure 24: Joint limit forcing, selectively damped least squares.

5.3.5 Gradient projection

With the pseudoinverse method, we find the solution of the least-squares problem. Nevertheless, when having redundant degrees of freedom, we can optimize according to another criterion, and one way of doing so without altering much the zero-error tracking is to create a secondary task as a gradient of a function, and projecting to the kernel of the primary task.

When we have a redundant manipulator, we can add a secondary objective to the solution of the inverse kinematics. An easy way of doing so is to create a cost function H, calculate its gradient ∇H, and project it to the kernel of the matrix J. Why projecting into the kernel of J? Knowing that for any position of the manipulator, $\dot{x} = J \cdot \dot{q}$, this means that if $\dot{\theta} \in ker(J)$, then $\dot{x} = 0$, so it would not affect the error. In practice, as the step is not infinitely small, the linearization done by projecting the vector to the nullspace would indeed generate some additional error.

This projection can be done multiplying any vector v by the matrix $P = (I - J^{\dagger} \cdot J)$.
Proof: if $P \cdot v = (I - J^{\dagger} \cdot J) \cdot v \in ker(J)$, $\forall v \in \mathbb{R}^7$ then $J \cdot (P \cdot v) = 0$.
so $J \cdot (P \cdot v) = J \cdot (I - J^{\dagger} \cdot J) \cdot v = (J - J \cdot J^{\dagger} \cdot J) \cdot v = (J - J) \cdot v = 0$, $\forall v \in \mathbb{R}^7$.
And this method would be expressed as:

$$\Delta\theta = \alpha \cdot J^{\dagger} \cdot e + \mu(I - J^{\dagger} \cdot J) \cdot \nabla H, \tag{20}$$

with μ a scalar indicating the magnitude of the projection, and ∇H the vector to project.

And in this case, the reconstruction error at the differential level, for $\alpha = 1$ is:
$\dot{x}_e - J \cdot \dot{\theta} = \sum_{i=r+1}^{m} \frac{\mathbf{u}_i^T \cdot \dot{\mathbf{x}}_e}{\sigma_i} \mathbf{v}_i + J \cdot \sum_{i=r+1}^{m} (\mathbf{v}_i^T \nabla \dot{H}) \cdot \mathbf{v}_i = \sum_{i=1}^{r} \frac{\mathbf{u}_i^T \cdot \dot{\mathbf{x}}_e}{\sigma_i} \mathbf{v}_i$, if $\nabla \dot{H} \in Ker(J)$.

Now, after some experimentation, we have seen that the gradient projection method pushes the joints away from their limits, but its effect is hierarchically under the 0-error tracking, and so it does not effectively avoid joint limits.

So we will experiment with the manipulability gradient (∇M) and a joint-centering function $H = -\lambda\theta$ projected onto the kernel of the Jacobian matrix, this would, supposedly, solve the stucking problem on singularities. Using:

$\Delta\theta = \alpha J^{\dagger} e + \mu(I - J^{\dagger} J)\nabla M$, being $\mu = 0.2$ and α with a small value (the least eigenvalue of matrix J) in the cases where singularity-avoidance is more relevant, or $\Delta\theta = \alpha J^{\dagger} e + \mu(I - J^{\dagger} J)\nabla H$, with H the joint-centering function otherwise.

- Vertical singular end position using M.
  Computation time of 0.0914s.As we can see in Figure 25, the method converges fastly as without the kernel projection (Figure 14).
- Joint limit forcing. With a computation time of 0.3307s, we now show the effect of projecting the gradient of the H function (Figure 26) which, theorically, pushes the joints away from their limits. We can see that, even with a small step, the fact that the joint limits are only treated on the kernel of the main task is not enough to ensure those are avoided.
- Horizontal end position. we can now see in Figure 27 (with a computation time of 0.1021s) that the gradient of the manipulability (∇M) has overcome the problem of getting stuck in the initial singularity.

5.3.6 Task priority

Task priority algorithms are a generalization of the gradient projection methods. The main idea is to have an ordered set of tasks, and project each task onto the kernel of the previous tasks

When dealing with redundant degrees of freedom, sometimes we have many tasks to perform. One example would be a humanoid robot trying to reach an object with its hand, respecting its

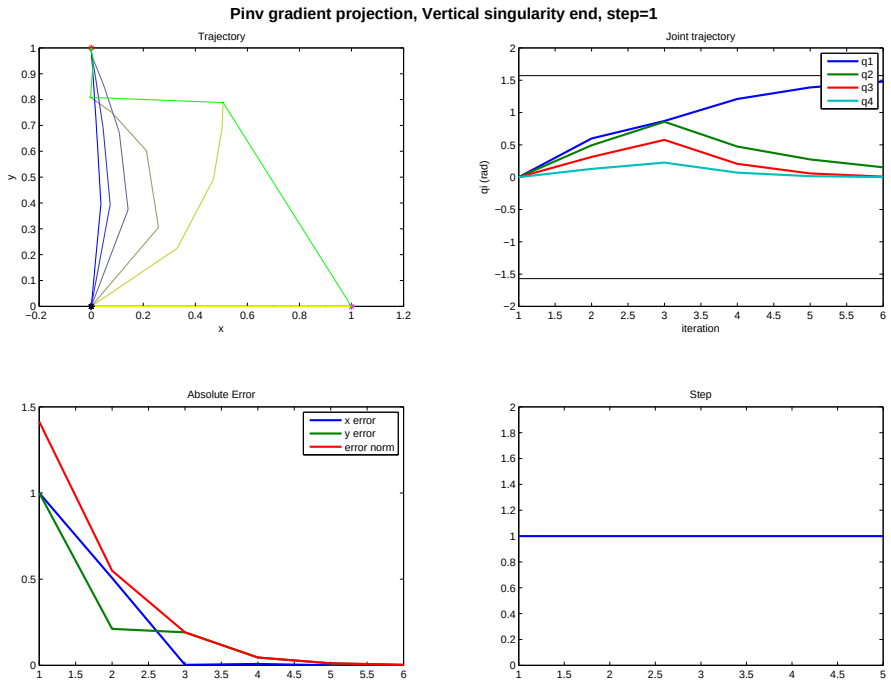

Figure 25: Gradient projection with pseudoinverse method. Vertical singular goal.

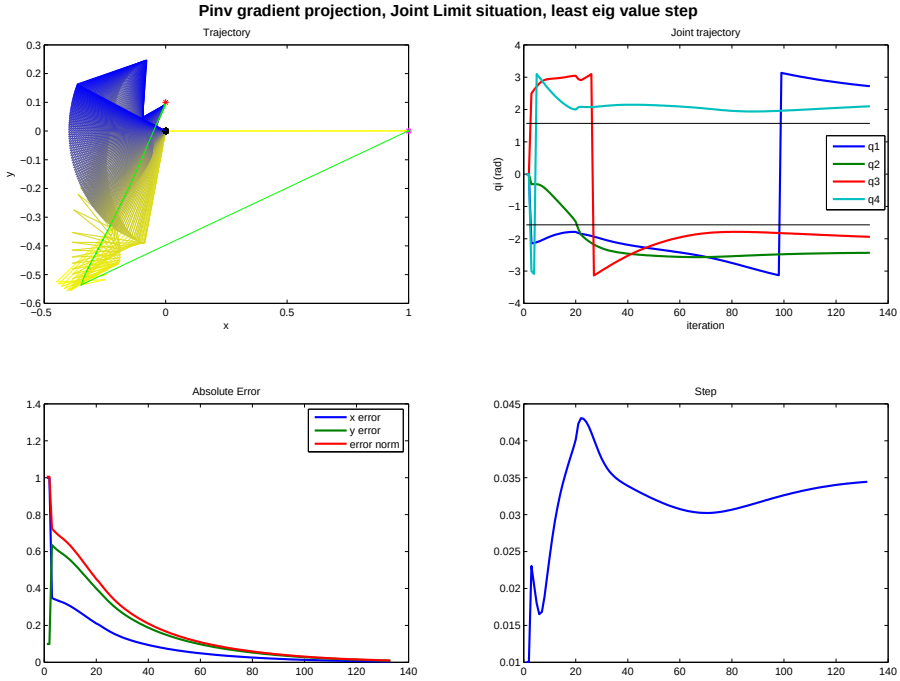

Figure 26: Joint limit forcing, gradient projection with pseudoinverse method.

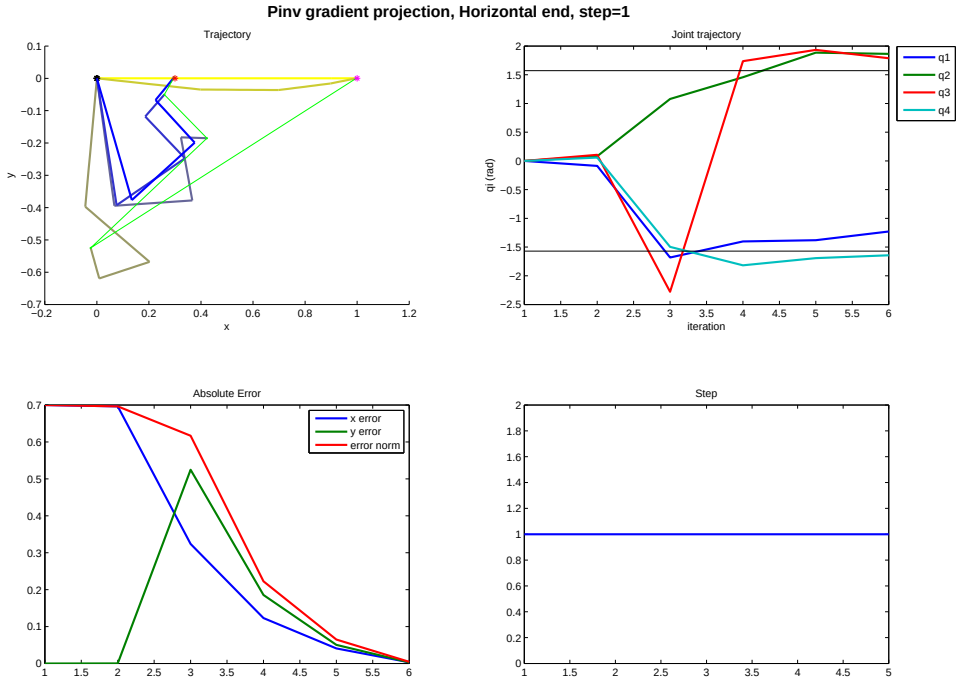

Figure 27: Gradient projection on pseudoinverse method

joint limits and keeping its gravity center on a region to maintain equilibrium. This example illustrates that sometimes there are tasks that are more important than reaching a position. We do not care if we can reach an object if the robot will fall because it did not keep its equilibrium.

For this kind of purposes, Siciliano and Slotine [19] generated a method to prioritisize a list of tasks, in a defined hierarchical order. The idea is to define a set of tasks $t_1, ..., t_k$, and their errors $e_1, ...e_k$. Then to compute a *Jacobian matrix* for each task, and perform task 1, then project task 2 into its kernel, and project task 3 into the kernel of tasks 1 and 2, etc.

So let e_1 be the error of task 1, and let J_1 be its Jacobian $J1 = \frac{\partial e_1}{\partial \theta}$.

Then we calculate $\Delta\theta_1 = J_1^{\star} e_1$, where $\star$ is an inverse operator (commonly pseudoinverse). Then, for each i from 2 to the number of tasks k, we do

$$\Delta\theta_i = \Delta\theta_{i-1} + (J_i \cdot P_{i-1}^A)^{\star}(e_i - J_i \Delta\theta_{i-1}), \tag{21}$$

where $P_{i-1}^A = I - J_i^{A^{\star}} J_i^A$ is the kernel projection operator and J_i^A is an augmented Jacobian $J_k^A = \begin{bmatrix} J_1 \\ \vdots \\ J_k \end{bmatrix}$.

Typical application

Now an application of this structure would be to generate an algorithm where the first task (most priority) is to avoid joint limits, and as a secondary task, the positioning goal.

We can define, for a robot, $J_1 = H_m$, m being the number of joints of the manipulator, and matrix H_m defined with the function (28) as

$$H_m = diag(1 - h_\beta(\theta_i)). \tag{22}$$

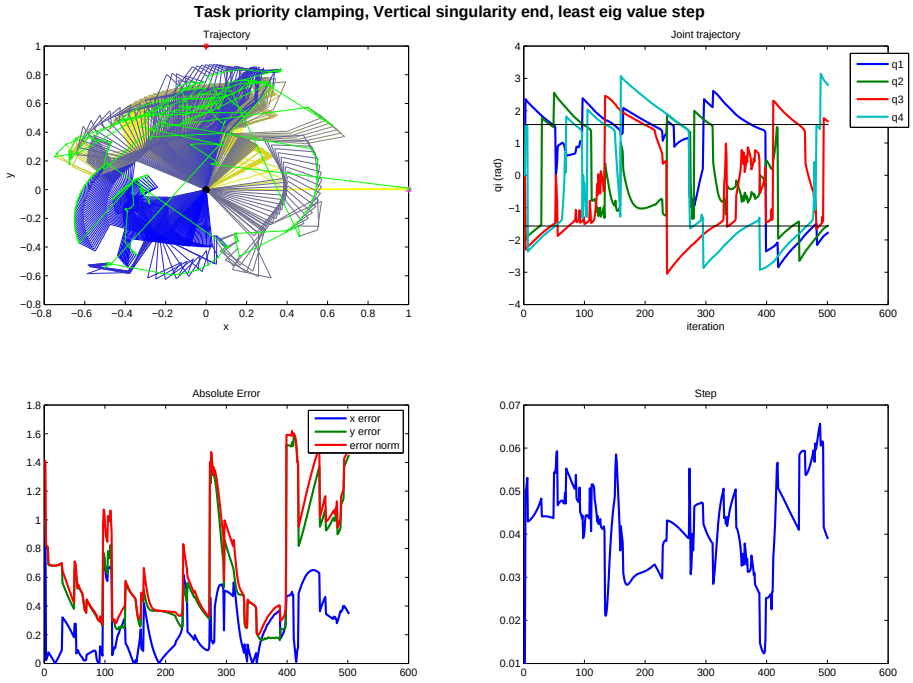

Figure 28: Task Priority method.

By observing Figure 30 this means that, as approaching ith joint limit, the ith value of the diagonal of H_m becomes smaller, and 0 at the limit, blocking the joint.

We also define the main task error as $e_1 = -\lambda_{JL} \cdot \theta$, being λ_{JL} a scalar to weight the importance of joint limits. Typically,$\lambda_{JL} = 0.1 \div 0.5$

With this definition of e_1 and J_1, $\Delta\theta_1$ will be zero when the joints' positions are away from their limits (at a distance β for each joint), and will have a *push-to-center* value when in the limits neighbourhood. On its kernel (i.e.; the joints which are not forced to move to the center due to their proximity to the limit), it is applied a second task to reach the goal with:

$J_2 = J$ the Jacobian matrix of the robot.

$e_2 = x_d - x$ the positioning error.

And then the method following a task-priority hierarchy is:

$$\Delta\theta = J_1 e_1 + J_2 (I_m - J_1^{\dagger} J_1)^{\dagger} (e_1 - J_2 J_1 e_1) \tag{23}$$

Now we can see a clear example of what has been exposed about the continuity of the pseudoinverse operator:

- Vertical singular end position. Now the discontinuity on the operator † triggers the activation of the secondary task (which is to reach the goal) as seen in Figure 28. This makes a very variable dimension of the joints vector space that tries to achieve the goal (i.e., the kernel of the main task), causing many oscillations, even for a small step.

5.3.7 Jacobian Weighting Methods

Apart from gradient projection methods, there are other ways to perform optimization of the redundant degrees of freedom, or avoiding joint limits. One of them is to penalize the motion

of some of the joints, depending on a weighting matrix. This weighting matrix will modify each joint's response according to a specified criterion.

In [3], Chan and Dubey define the $\Delta\theta$ which minimizes

$$\|\theta\|_W = \sqrt{\theta^T W \theta}, \tag{24}$$

where W is an $m \times m$ diagonal matrix reflecting a weight on each joint, depending on its relevance (according to a specified criteria) instead of the common euclidean norm $\|\theta\| = \sqrt{\theta^T\theta}$.

To this purpose, a new set of joints can be defined as

$$\theta_W = W^{1/2}\theta, \tag{25}$$

then

$$\dot{x} = J\dot{\theta} = JW^{-1/2}W^{1/2}\dot{\theta} = JW^{-1/2}\dot{\theta}_W = J_W\dot{\theta}_W \tag{26}$$

and we can compute the pseudoinverse of J_W

$J_W^\dagger = J_W^T(J_W J_W^T)^{-1} = W^{-1/2}J^T(JW^{-1/2}W^{-1/2}J^T)^{-1} = W^{-1/2}J^T(JW^{-1}J^T)^{-1}$
And if $\dot{\theta}_W = J_W^\dagger\dot{x}$, we have $\dot{\theta} = W^{-1}J^T(JW^{-1}J^T)^{-1}\dot{x}$.

This pseudoinverse matrix of J_W can also be damped or filtered, but then always premultiplied by $W^{-1/2}$, which comes from the change of θ_W to a joint change on θ.

The influence of $\omega_i = W_{i,i}$ is that, the greater the ω_i, the less θ_i will vary in the step.

With this structure created, one can take $W = diag(\omega_1, ...\omega_m)$, where $\omega_i = 1 + |\frac{\partial H}{\partial\theta_i}|$, being $H : Q \subseteq \mathbb{R}^m \to \mathbb{R}$ a scalar value to minimize. A typical choice of H is like in eq. (6). What translates into:

$\frac{\partial H}{\partial\theta_i} = \frac{(\theta_i^{max}-\theta_i^{min})^2(2\theta_i-\theta_u i^{max}-\theta_i^{min})}{2m(\theta_i^{max}-\theta_i)^2(\theta_i-\theta_i^{min})^2}$
Nevertheless, in [3] it is pointed out that this gradient does not give information about whether a joint is moving towards its limit or away from it, so it might be reducing a movement that, not only is valid, but also sends the joint to a more *central* position. To this purpose, in [3] it is defined, for two consecutive steps k and k+1 $\Delta H_{i_K} = |\frac{\partial H}{\partial\theta_i}|_k - |\frac{\partial H}{\partial\theta_i}|_{k-1}$ and redefine

$$\omega_{i_K} = \begin{cases} 1 + |\frac{\partial H}{\partial\theta_i}| & \text{if } \Delta H_{i_K} \geq 0 \\ 1 & \text{if } \Delta H_{i_K} < 0 \end{cases}$$

With this new weight, if the robot is moving towards a joint limit, the joint is weighted to reduce its influence and so, avoid it. And it moves away from a joint limit, the joint is not affected by a weight.

To perform some examples, the case of a vertical singular end position presents no problems, while the horizontal end position does have the problem as in the Jacobian transposed or pseudoinverse methods. Now when forcing a limit:

- Joint limit forcing. Figure 29 (computation time of 0.0622s) we can see the behaviour is better than with the gradient projection method, and it does push joints back to their range, although it does not achieve such task 100% efficiently.

5.3.8 Joint Clamping

An intuitive way of avoiding to surpass a joint limit, is to block the value of the joint trying to surpass its limit. We call this clamping a joint.

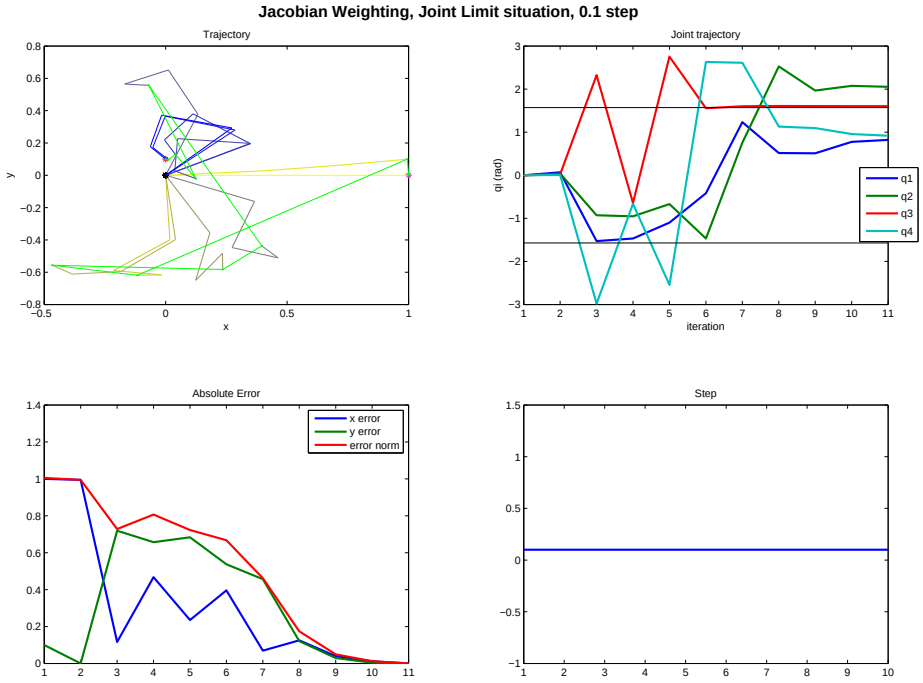

Figure 29: Activation of a joint depending on its normalised value between -1 and 1 using a $\beta = 0.2$ threshold.

In [17], a method is defined by adding a term when updating the joint state: $\theta = \theta + H \cdot \Delta\theta$, instead of $\theta = \theta + \Delta\theta$, where H can be a diagonal matrix, with: $H_{i,i} = \begin{cases} 0 & \text{if } q_i > q_i^{max} \text{ or } q_i < q_i^{min} \\ 1 & \text{otherwise} \end{cases}$

This control law clamps any motion that violates joint limits, but it does not push the joint away from them. In this way, when the robot reaches a joint limit, it looses this degree of freedom, and goes on with the others to reach the target. Nevertheless, when a joint is clamped, a discontinuity in the movement occurs, which we will see.

Another way of representing the clamping of joints, is to treat it as if it was a weighting matrix, and then define the method as:

$$H\Delta\theta = \alpha H(JH)^{\dagger} e. \tag{27}$$

In fact, in a task-priority based method, some degrees of freedom can be blocked in some tasks, to be used in others tasks. When this happens, we can also analyse whether this method is continuous with respect to the activation matrix H or not. If we want $\alpha H(JH)^{\dagger}$ to be continuous, the following parts must be continuous with respect to H:

- **The step** α, will not usually depend on H, so α will be continuous with respect to H.
- **The matrix** JH. As, by construction, J is a constant w.r.t H, it is continuous.
  Now for JH to be continuous, it is only needed that H is continuous. To this purpose, an alternative of H that is continuous can be defined by setting a threshold β to the upper and lower joint limits, use a continuous function that evolves from 0 at a distance β from the joint limit to 1 on the joint limit, such as the one proposed in [12].
  $\beta_i = \beta \cdot (q_i^{max} - q_i^{min})$
  and then

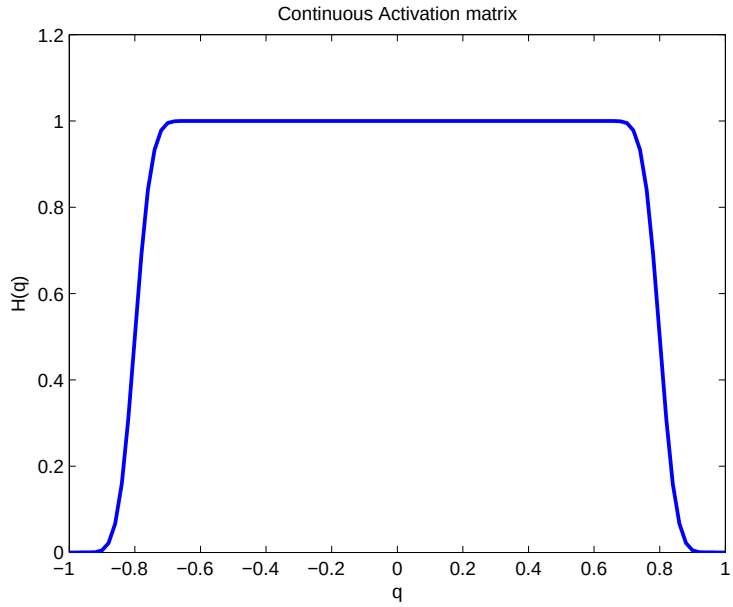

Figure 30: Deactivation of a joint close to its limit.

$$H_{i,i} = \begin{cases} 1 & \text{if } q_i > q_i^{max} \text{ or } q_i < q_i^{min} \\ f_{\beta_i}(\beta_i + q_i^{min} - q_i) & \text{if } q_i \in [q_i^{min}, q_i^{min} + \beta_i] \\ f_{\beta_i}(\beta_i - q_i^{max} + q_i) & \text{if } q_i \in [q_i^{max} - \beta_i, q_i^{max}] \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

where $\forall x \in [0, \beta]$, we define $f_{\beta_i}(x) = \frac{1}{2}(1 - tanh(\frac{1}{1-x/\beta_i} - \frac{\beta_i}{x}))$
This new matrix H gives a continuous clamping behaviour as in Figure 30, which ensures continuity of JH w.r.t. H.

- **The matrix** H is also continuous with the same criteria as for JH.
- **the operator** †.
  We already know this operator is continuous with respect to J when its rank is constant. Otherwise, the pseudoinverse operator presents a discontinuity:
  As seen before, $J^\dagger = \sum_{i=1}^{m} \frac{1}{\sigma_i}\mathbf{u}_i\mathbf{v}_i^T = \sum_{i=1}^{r} \frac{1}{\sigma_i}\mathbf{u}_i\mathbf{v}_i^T$, where r is the rank of matrix J. Then we have that, $\lim_{\sigma_j \to 0} \frac{1}{\sigma_j} = \infty$, but when a singular value of the matrix J equals zero, by construction of the pseudoinverse matrix, the eigenvalue 0 is taken, so the pseudoinverse operator is discontinuous when matrix J looses rank. This can be avoided by damping or filtering the Jacobian matrix as we have seen in the previous sections.
  When looking to the continuity of the operator † with respect to H, we will prove that it is not continuous: let's define a joint activation matrix as $H = \begin{pmatrix} h & 0 \\ 0 & I \end{pmatrix}$, with h a scalar and I the $(m-1) \times (m-1)$ identity matrix.
  Now, we will check the continuity of pseudoinverse operator at $h = 0$. To do so, it is great help the *Theorem 4.2* in [13], which states that, $\Delta\theta = (GJ)^\dagger Ge$, being G a diagonal activation matrix, then it is:

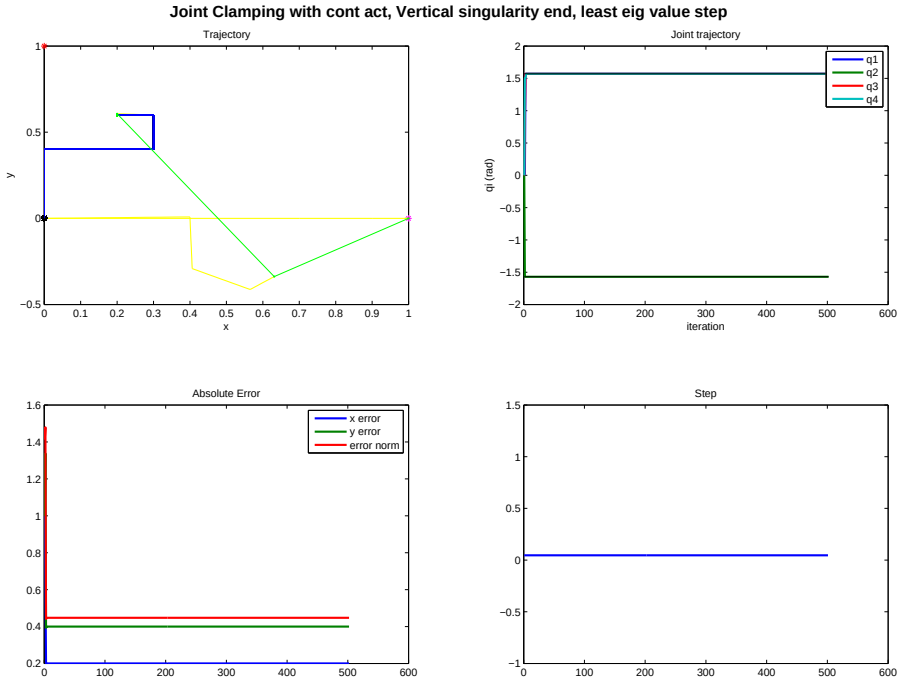

Figure 31: Clamping joints.

$(GJ)^\dagger Ge = (\hat{G}J)^\dagger \hat{G}e$, where $\hat{G}$ is a different diagonal activation matrix, with the only restriction that $\hat{G}_i = 0 \Leftrightarrow G_i = 0$, $\forall i$. This is equivalent to say $H(JH)^\dagger e = \hat{H}(J\hat{H})^\dagger e$ (equation (27)) for any matrix $\hat{H}$ verifying $\hat{H}_i = 0 \Leftrightarrow H_i = 0$, $\forall i$.
In words, it means that *the effect of nonzero diagonal elements on G does not depend on their values, if they are not zero.*
The same theorem is valid for the case of restricting joints, with method $\Delta\theta = H(JH)^\dagger e$ so we have

$$lim_{h\to 0} \begin{pmatrix} h & 0 \\ 0 & I \end{pmatrix} \left[J \begin{pmatrix} h & 0 \\ 0 & I \end{pmatrix} \right]^\dagger = lim_{h\to 0} \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix} \left[J \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix} \right]^\dagger =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix} \left[J \begin{pmatrix} 1 & 0 \\ 0 & I \end{pmatrix} \right]^\dagger = J^\dagger$$

as the effect of $h_{i,i} \neq 0$ is the same as $h_{i,i} = 1$.
On the other hand, we have that, for $h = 0$, the first column of JH will be zero, and so the first row of $(JH)^\dagger$, thus the operator is discontinuous at $h = 0$.

Now, using the method as $\Delta\theta = \alpha H(JH)^\dagger e$, we can see it gets stuck easily because of this clamping, even with the continuous clamping function.

- Vertical singular end position. As we can see in Figure 31, at the first step one joint reaches its maximum. Then this joint is blocked and does not move again on the following iterations, the robot being unable to reach the goal. This evidentiates that clamping is not enough to reach the target while avoiding joints, these have to be pushed away from joint limits.

5.3.9 Continuous Clamping

To solve the discontinuity on the activation/deactivation of joints, a new inverse operator has been created to assure the continuity of the method when the activation of tasks or joint is triggered. These inverse operator will be defined so as to take the value of the Moore-Penrose pseudoinverse when joints are fully active or blocked, and with this inverse operators a continuous task-priority method can be used. For practical reasons, we recommend reading [12] to better understand the definition of this inverse

As we have seen, the pseudoinverse operator is not continuous when dealing with activation matrices or task priority. To this purpose, as introduced, [14] and [13] propose a continuous w.r.t activation matrices inverse, based on a diagonal activation matrix to clamp tasks, and also joints on an algorithm.

- **Task Clamping**. The original purpose of [14] and [13] is to create a control scheme in which tasks may vary its activation, named *varying-feature-set tasks*, which are tasks with the form $e_q = Ge$, G being a diagonal continuous matrix with its values in the interval $[0, 1]$. When $g_i = G_{i,i} = 1$, the task is active. On the other hand, when $g_i = G_{i,i} = 0$, task i becomes inactive.
  An example for this would be, in an oriented spatial robot (where the task space has 6 degrees of freedom, 3 for position and 3 for orientation), to activate the orientation tracking on the Jacobian matrix of a manipulatori only when the robot is close enough to the goal. Then, matrix G would be:
  $G = \begin{pmatrix} I_3 & 0_{3\times3} \\ 0_{3\times3} & A \end{pmatrix}$, where

$$A = \begin{Bmatrix} 0_{3\times3} & \text{if } e > e_{min} - \epsilon \\ f(e, e_{min}, \epsilon) \cdot I_3 & \text{if } e_{min} > e > e_{min} - \epsilon \\ I_3 & \text{if } e < e_{min} \end{Bmatrix},$$

  being $f(e, e_{min}, \epsilon)$ a smooth function as in equation (28).

  To this purpose, a control method as $\Delta\theta = (GJ)^\dagger Ge$ could be used, but it would have the continuity problem as previously commented. To solve it, a new inverse operator is defined in these articles, as

$$\begin{gathered} \oplus : M_{n\times m}(\mathbb{R}) \times M_{n\times n}(\mathbb{R}) \to M_{m\times n}(\mathbb{R}) \\ (A, G) \mapsto A^{\oplus G} \end{gathered}$$

  to verify:
  - Given a matrix A, $\oplus$ is continuous with respect to G.
  That is to avoid the discontinuities caused by the change of task activation on matrix G.
  - if $\forall i = 1..n, g_i \in \{0, 1\}$, then $A^{\oplus G} = (GA)^\dagger = (GA)^\dagger G$
  This means that when matrix G is a binary diagonal matrix, the operator must have the same values as the Moore-Penrose pseudoinverse. This is to keep consistence with the standard pseudoinverse.

  With these specifications, a set of auxiliar matrices are defined: *The Coupling Matrices of a Matrix J* are $n \times m$ matrices indexed by the elements of the power set $\wp$ (the set of subsets) of $N = \{1, ..., n\}$
  if $P = \varnothing$, $X_\varnothing = 0_{m\times n}$
  else $\forall P \in \wp(N)$, $X_P = J_P^\dagger - \sum_{Q \subsetneq P} X_Q$

where $J_P = G_0 J$, being G_0 a binary diagonal matrix with $G_{0_i} = 1$ if $i \in P$ and 0 otherwise.

With these coupling matrices, the *Continuous Inverse of a Matrix J with task activation given by G* is defined as:

$$J^{\oplus G} = \sum_{P \in \wp(N)} (\prod_{i \in P} g_i) X_P = \sum_{P \in \wp(N)} (\prod_{i \in P} g_i)(\prod_{i \notin P} (1 - g_i)) J_P^{\dagger} \tag{29}$$

And it is proven, again in [14], that this pseudoinverse operator verifies the two properties required.

Also, in [12] this pseudoinverse is generalised for the case of a non-diagonal activation matrix, say W. If we use the singular value decomposition of W, then $W = USU^T$ and $(WJ)^{\dagger}W = (SU^TJ)^{\dagger}SU^T$, and the *Continuous Inverse if a Matrix J with task activation given by a non-diagonal matrix W* is

$$J^{\oplus W} = J_U^{\oplus S} U^T = \sum_{P \in \wp(N)} \left(\prod_{i \in P} \sigma_i \right) X_P^{J_U} U^T, \tag{30}$$

where $X_P^{J_U}$ are the task-coupling matrices of $J_U = U^T J$.

- **Joint Clamping**. As seen before, what we expect to do is to restrict a joint movement to avoid its limits in spite of a task. To this purpose, our starting control scheme would have the form
$\Delta\theta = H(JH)^{\dagger}e$, but defining an operator

$$\begin{gathered} \oplus : M_{n\times m}(\mathbb{R}) \times M_{m\times m}(\mathbb{R}) \to M_{m\times n}(\mathbb{R}) \\ (A, H) \mapsto A^{H\oplus} \end{gathered}$$

we can calculate the adapted *Continuous Inverse of a Matrix J with joint activation given by H* as:

$$J^{H\oplus} = \sum_{Q \in \wp(M)} (\prod_{i \in Q} h_i) Y_Q = \sum_{Q \in \wp(M)} (\prod_{i \in Q} h_i)(\prod_{i \notin Q} (1 - h_i)) J_Q^{\dagger} \tag{31}$$

With its coupling matrices now defined as
if $Q = \varnothing$, $X_{\varnothing} = 0_{mxn}$
else $\forall Q \in \wp(M)$, $Y_Q = J_Q^{\dagger} - \sum_{R \subsetneq Q} Y_R$
where $J_Q = JH_0$, being H_0 a binary diagonal matrix with $H_{0_i} = 1$ if $i \in Q$ and 0 otherwise.

This matrix is also defined in [12] as a *left-activated continuous inverse* from the task-clamping continuous inverse:
$J^{H\oplus} = ((J^T)^{\oplus H})^T$
Note that, then it is $((J^T)^{\oplus H})^T = \sum_{Q \in \wp(M)} (\prod_{i \in Q} h_i)(\prod_{i \notin Q} (1 - h_i)) Y_Q$ (i.e.: both definitions match)
Proof
By definition, we have that $((J^T)^{\oplus H})^T = \sum_{P \in \wp(N)} (\prod_{i \in P} g_i)(\prod_{i \notin P} (1 - g_i)) X_P^T$, being X_P the taks activation matrices of J^T.

It is then enough to demonstrate that $X_P^T = Y_P$, for a diagonal activation matrix H, and using J^T for computing X_P. And this demonstration can be reduced to view $((H_0 J^T)^\dagger)^T = (JH_0)^\dagger$:
Consider a full rank matrix J (the generalization for degenerate matrices is trivial). Then we have, by definition of the *Moore-Penrose pseudoinverse*:
for an $m \times n$ matrix, the pseudoinverse of a full ranked matrix is defined as
$(A^T A)^{-1} A^T$ if $m > n$,
$A^T (AA^T)^{-1}$ if $m < n$.
Then it is $(H_0 J^T)^\dagger = (H_0 J^T J H_0^T)^{-1} (H_0 J^T)^T$
As H_0 is diagonal, it is $H_0^T = H_0$ and $H_0 J = J H_0$, thus:
$\left((H_0 J^T)^\dagger\right)^T = \left((H_0 J^T J H_0^T)^{-1} (H_0 J^T)^T\right)^T = (H_0 J^T) \left((J^T H_0 H_0^T J)^{-1}\right)^T =$
$= (JH_0)^T (JH_0 H_0^T J^T)^{-1} = (JH_0)^\dagger$ ■

By analogy to the previous case, a non-diagonal joint activation matrix $D = V \Sigma V^T$ can be used, and so

$$J^{D\oplus} = J_V^{\Sigma\oplus} = \sum_{Q \in \wp(M)} \left(\prod_{i \in Q} \sigma_i \right) V Y_Q^{J_V}, \tag{32}$$

being $Y_Q^{J_V}$ the coupling matrices of $J_V = VJ$.

On [12], these inverse operators are combined with the task-priority method used as in (23), generate the following algorithm: $\Delta\theta_1 = J_1^{\oplus H} e_1$, being H the same matrix as defined in (22)
And then the following tasks defined as:

$$\Delta\theta_i = \Delta\theta_{i-1} + J_i^{P_\oplus^{i-1}\oplus}(e_i - J_i \Delta\theta_{i-1}), \tag{33}$$

where $P_\oplus^0 = I$, and $P_\oplus^1 = P_\oplus^{i-1} - J^{P_\oplus^{i-1}\oplus} J_i$ is the *kernel projection* operator.
In [12] it is demonstrated that the projector operator $P_\oplus^{i-1}\oplus$ makes the secondary task not to affect the primary, i.e.:
$J_{i-1} J_{i+1}^{P_\oplus^{i-1}\oplus} = 0$
Now we show the results for this method, with the task order as in (23):

$$\Delta\theta = \Delta\theta_1 + J_2^{P_\oplus^1\oplus}(e_2 - J_2 \Delta\theta_1) \tag{34}$$

With $\Delta\theta_1 = J_1^{H\oplus} e_1 = I_m^{H\oplus}(-\lambda_{jl}\theta) = H(-\lambda_{jl}\theta)$, as $J_1 = I_m$ and $e_1 = -\lambda_{jl}\theta$ so we finally have:

$$\Delta\theta = H(-\lambda_{jl}\theta) + J_2^{(I_m - H)\oplus}(e_2 - J_2 H(-\lambda_{jl}\theta))$$

– Vertical singular end position. As we can see in Figure 32, this solution respects the joint limits. When a joint limit is surpassed, it is forced back to a feasible joint configuration. What arises now is that, as the goal position has only one solution $\theta = \begin{pmatrix} \pi/2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, with coordinate 1 being a joint limit. This solution is not fully reached due to the priority to avoid joint limits as the first task. There is an equilibrium between the primary task: pushing away the first joint from a value of $\pi/2$, and the secondary task, pushing its value to $\pi/2$. Nevertheless, the solution is approached correctly, although on the first step, the initial singularity causes a big step.

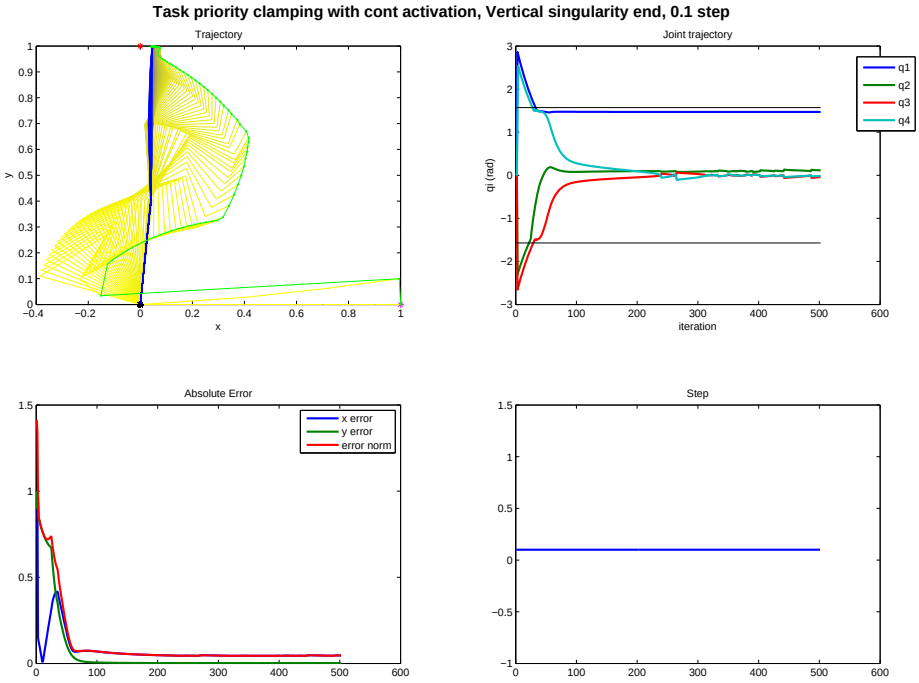

Figure 32: Continuous clamping task priority algorithm.

– Horizontal end position. For this case, as could be expected, the robot won't move. This is because it has been assured a continuity depending on the activation matrix, but no filtering or similar has been applied to solve singularities.
– Joint limit forcing. Now we could expect a correct behaviour of the robot when trying to force its limits. In spite of this, what we find is in Figure 33. This is due to the fact that the solution to this desired goal does not exist, so the algorithm keeps looking for an equilibrium point, that can only be an algorithmic singularity.

5.4 Proposed methods

Apart from what we have been exposing, now we propose two new methods. The first one, is a new way of filtering the Jacobian matrix, so as to effectively avoid large gains and large condition numbers of the Jacobian matrix, while the second one is a mixture of the continuous task-priority clamping algorithm with the selective damping, to find a very smooth algorithm, as we will see.

5.4.1 Smooth eigenvalue filtering.

Although it may seem that the Jacobian damping and filtering would solve any problem when a singularity occurs, if we analise the behaviour of the function

$$f_\lambda : \mathbb{R} \to \mathbb{R}$$
$$\sigma \mapsto f_\lambda(\sigma) = \frac{\sigma}{\sigma^2+\lambda^2}$$

we can see that its derivative w.r.t σ is
$\frac{\partial f_\lambda}{\partial \sigma} = \frac{\lambda^2-\sigma^2}{(\sigma^2+\lambda^2)^2} = 0 \Leftrightarrow \sigma = \pm\lambda$

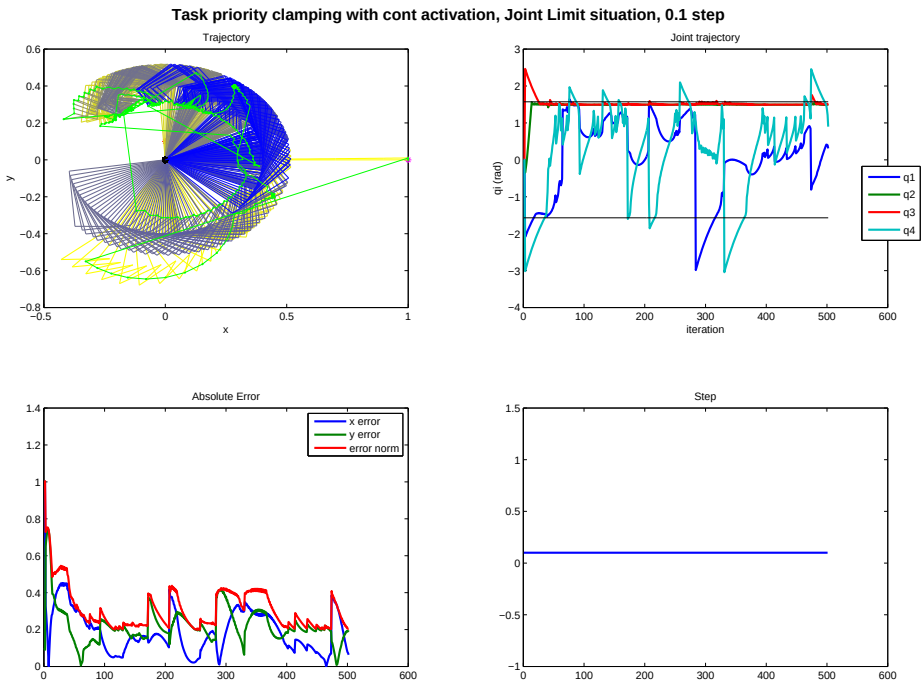

Figure 33: Continuous clamping task priority algorithm.

For $\lambda \neq 0$, we have $f_\lambda(\lambda) = \frac{1}{2\lambda}$. This means that, although this damping ensures continuity at $\sigma = 0$, for small values of σ close to λ, $f_\lambda(\sigma)$ takes very large values, which will not solve from a practical point of view the discontinuity of the singularity itself.

For this reason, we propose a new filtering of the Jacobian matrix, so if

$$J = \sum_{i=1}^{m} \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

we define

$$\hat{J} = \sum_{i=1}^{m} h_{\nu,\sigma_0}(\sigma_i) \mathbf{u}_i \mathbf{v}_i^T,$$

where $h_{\nu,\sigma_0}(\sigma_i) = \frac{\sigma^3+\nu\sigma^2+2\sigma+2\sigma_0}{\sigma^2+\nu\sigma+2}$ is our proposed rational function with:

For a suitable shape factor ν (it has been checked that a value of 10 works correctly).

It can be easily seen that $h_{\nu,\sigma_0}(\sigma_i)$ verifies:

- $h_{\nu,\sigma_0}(\sigma_i)$ is continuous and differentiable everywhere if $\lambda^2 < 8$, and for $\lambda^2 > 8$ it has two discontinuities, both for negative σ. So $h_{\nu,\sigma_0}(\sigma_i)$ is differentiable on the positive side of $\mathbb{R}$ which is where the eigenvalues of the Jacobian matrix are.
- $\lim_{\sigma\to 0} h_{\nu,\sigma_0}(\sigma_i) = \sigma_0$, $\forall\nu$, so σ_0 is the minimum value we will allow for the eigenvalues of the Jacobian matrix.
- $m = \lim_{\sigma\to\infty} \frac{h_{\nu,\sigma_0}(\sigma_i)}{\sigma} = 1$, $\forall\nu$ and $\forall\sigma_0$
  $n = \lim_{\sigma\to\infty} h_{\nu,\sigma_0}(\sigma_i) - m \cdot \sigma = 0$, so $h_{\nu,\sigma_0}(\sigma_i)$ has an asymptote of equation $y = \sigma$ for $\sigma \to \infty$.

So what we have is that $\hat{J}$ has lower-bounded eigenvalues and tends to J when its eigenvalues move away from 0, which is what we needed.

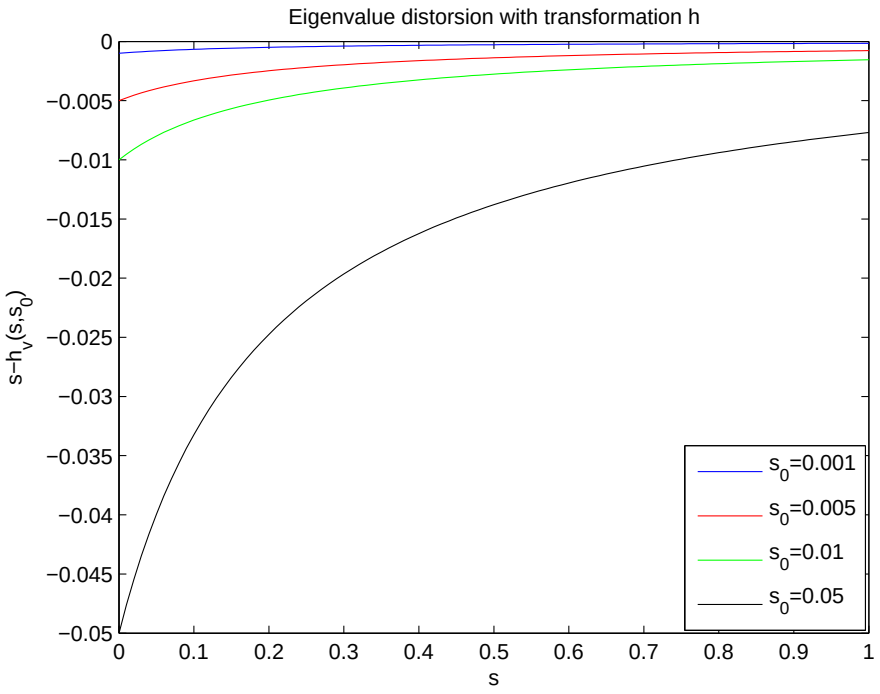

Figure 34: Eigenvalue distorsion when applying function h_ν

In addition, by construction, we have $ker(\hat{J}) = ker(J)$ when J is full-ranked, so secondary tasks on the kernel of J are not affected. Moreover, as the singularity sets are often a subspace of the joint space with zero measure, we only care about the neighbourhood of a singularity, more than about the exact singularity. The only case in which these kernels could be different is when starting at an exact singularity on a simulation. There it could be a first step in which those kernels are different. During the simulation, if the measure of the singularity set is zero, the probability of numerically being in an exact singularity drops to zero. In fact, reducing the dimension of the kernel of the Jacobian reduces, in this case, the probability of getting stuck in an algorithmic singularity (see Section 5.2).

The advantages of this method can be seen in Figure 35, where we plot the condition number of different methods when approaching a singularity in a 4R planar manipulator, for a damping factor of $\lambda = 2 \cdot 10^{-4}$, which is a strong damping, and allowing a maximum damping factor on the filtering algorithm (variable damping factor) as in (18) of $\lambda_{max} = 5\lambda$. As we have already commented, the pseudoinverse algorithm's condition number tends to infinity.

We can also see that the filtered and the damped Jacobian methods have an identical behaviour in the sense of the condition number of the Jacobian inverse, and they present a maximum at $\sigma = \lambda$, with a maximum condition number of almost 3000. This means that, although very close to 0, the condition number might be smaller than the proposed method on the region around $\sigma = \lambda$, the filtered or damped Jacobian methods only work well in this small region.

On the other side, the proposed method (with $\sigma_0 = 0.005$) presents a lower dependency on the lowest eigenvalue, when reaching the singularity.

We can also analise the behaviour of the condition number for different σ_0 values (Figure 36), where we can see that, even for a small value of σ_0, the algorithm is still stable.

Then we can compute $\hat{J}^\dagger = \sum_{i=1}^{r} \frac{1}{h_{\nu,\sigma_0}} \mathbf{v}_i \mathbf{u}_i^T$, being $r = rank(J)$ (assuming $\sigma_i > \sigma_{i+1}\ \forall i$), to use it instead of the common pseudoinverse.

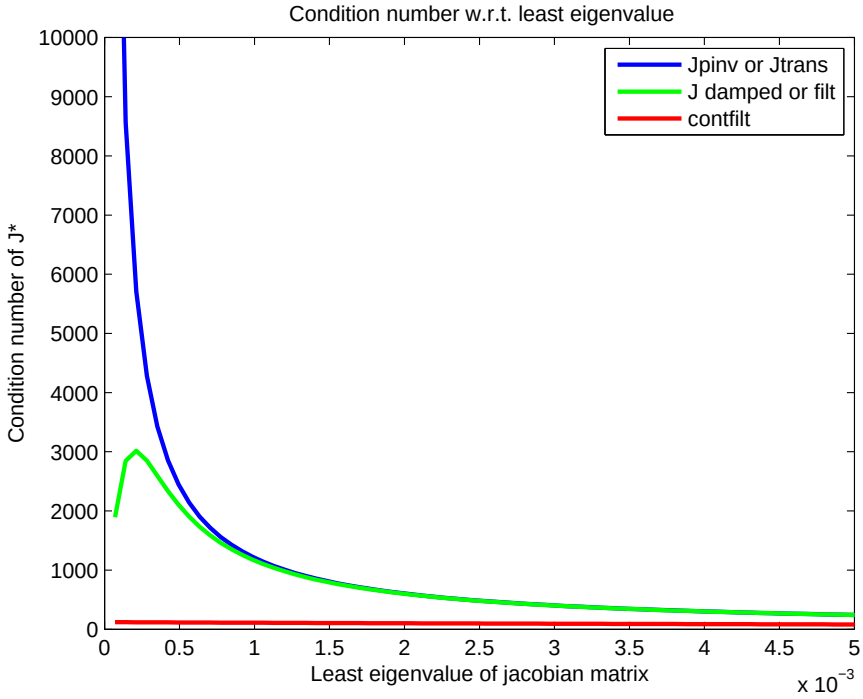

Figure 35: for $\sigma_0 < 0.01$ the difference between the eigenvalues becomes very small.

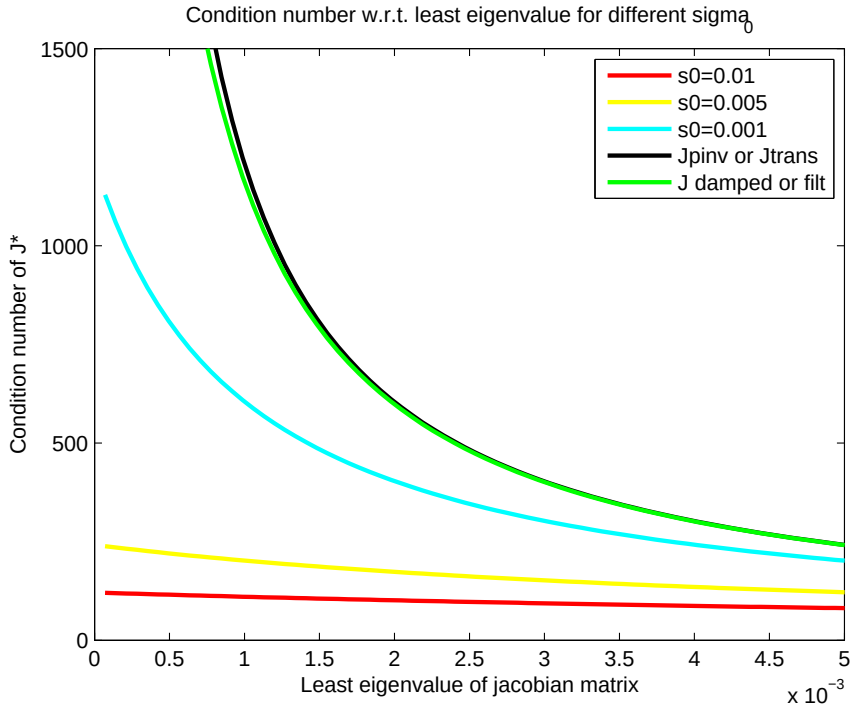

Figure 36: Ampliation of Figure 35, with different values of σ_0.

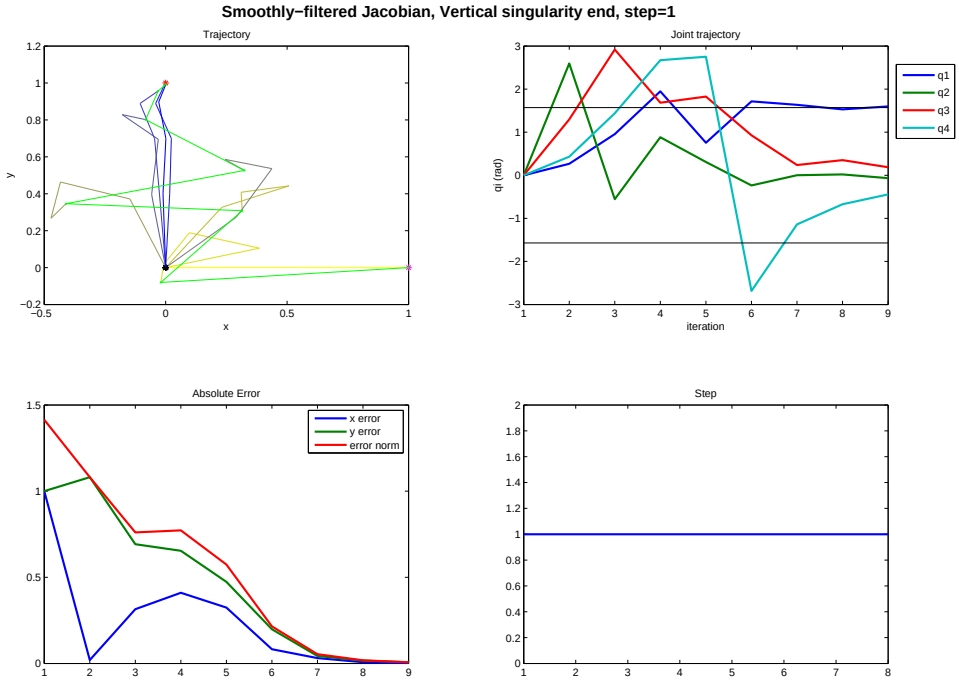

Figure 37: Smoothly-filtered Jacobian method. With a big step of $\alpha = 1$, the method converges fastly, although it keeps doing big steps.

Now, we can see the results when testing this method $\Delta\theta = \hat{J}^{\dagger}e$, with step $\alpha = 1$:

- Vertical singular end position. With a computation time of 0.0620s, this method converges even with big step $\alpha = 1$ (Figure 37)
- Horizontal end position. Even when the starting position is at a singularity, the method converges fastly (Figure 38) in a computation time of 0.0645s.
  In this case we can see in Figure 39 that, using a step of the least eigenvalue, the trajectory is very smooth, in the sense that it does not make leaps at the firsts steps.

5.4.2 Continuous Clamping combined with Selective Damping

Apart from smoothly filtering the Jacobian matrix, without altering it much, now we will pay attention to the algorithm of equation (33):

$$\Delta\theta_i = \Delta\theta_{i-1} + J_i^{P_{\oplus}^{i-1}\oplus}(e_i - J_i\Delta\theta_{i-1}),$$

and the version used in equation (34) (as $J_1 = I$ and, for the identity matrix, $I^{\oplus H} = H$. We can reorder the terms and split the position error-depending terms (e_2) from those which don't depend on it):

$$\Delta\theta = H(-\lambda_{jl}\theta) + J_2^{(I_m-H)\oplus}(e_2 - J_2H(-\lambda_{jl}\theta)) =$$

$$\Delta\theta = \left(I - J_2^{(I_m-H)\oplus}J_2\right)H(-\lambda_{jl}\theta) + J_2^{(I_m-H)\oplus}e_2 = \Delta\theta^{*1} + \Delta\theta^{*2} \tag{35}$$

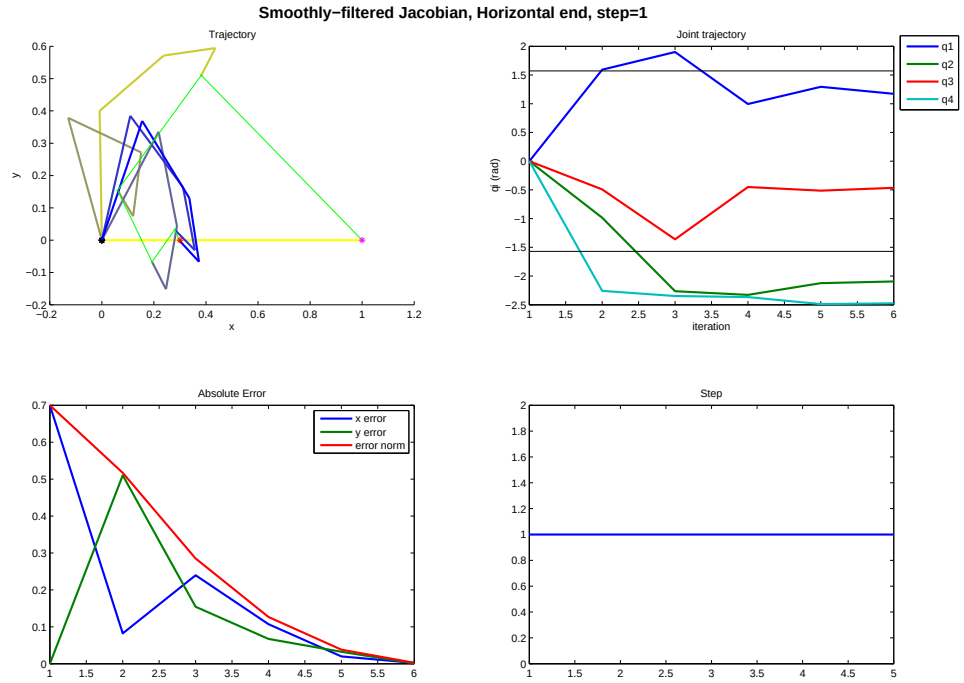

Figure 38: Smoothly-filtered Jacobian method. With the horizontal end position, the proposed method does not get stuck and converges in only 6 iterations..

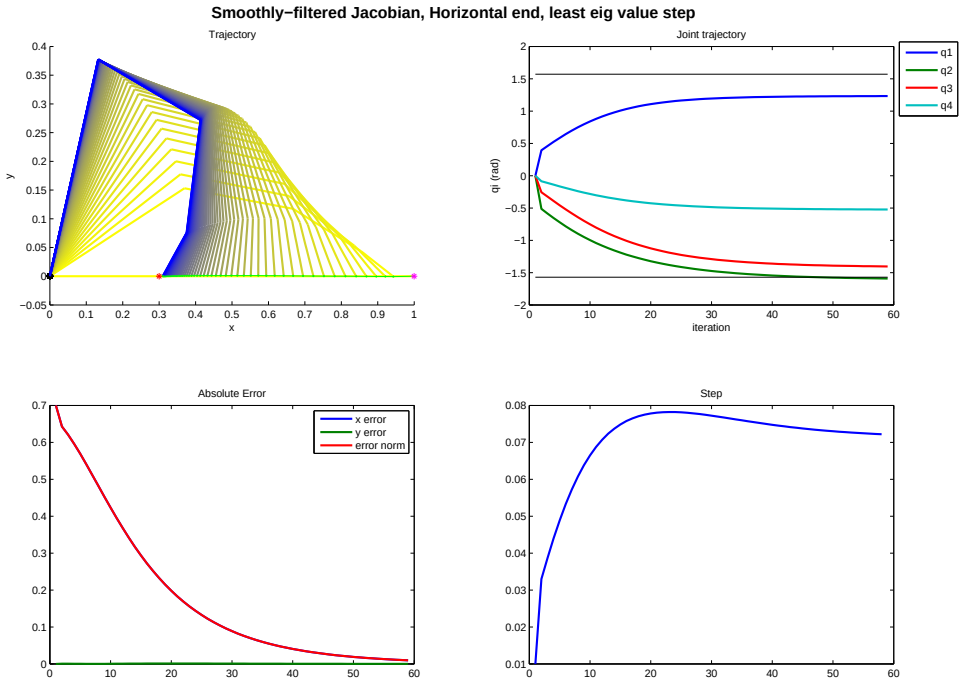

Figure 39: Smoothly-filtered Jacobian method

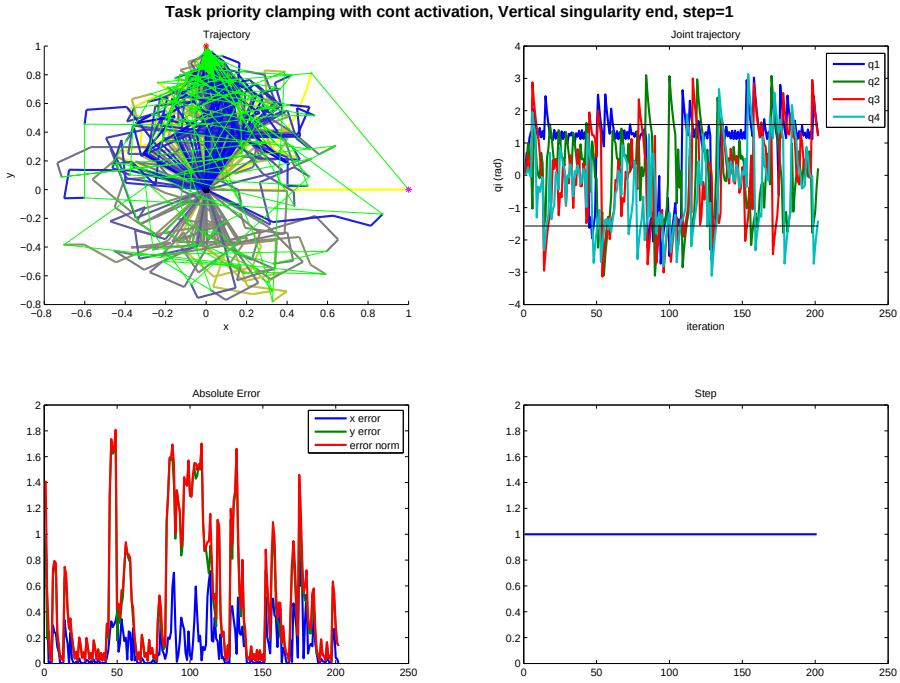

Figure 40: non smoothly-filtered Jacobian continuous clamping

This algorithm continuously avoids the joint limits, and the problems associated with the task clamping. Nevertheless, as we have seen in Figure 32, the first steps of the algorithm are large, and depending on the initial error, its behaviour can be a bit chaotic as in Figure 33.

For example, using the algorithm in equation (33) with a step of $\alpha = 1$ for the vertical singularity end, it gives an undesirable behaviour as seen in Figure 40, and smoothly filtering the Jacobian reduces this behaviour making it converge faster, but it does not solve it, as in Figure 41

To improve this behaviour, what we intend is to apply the ideas of the *Selectively damped least squares* method explained in Section 5.3.4. We will damp selectively each one of the task space eigenvectors of the Jacobian matrix J_2, taking care of the dependency of the position variation $J_2\Delta\theta$ with respect to the position error e_2.

To do so, we have to find a bound for $J_2\Delta\theta$, i.e.: the position variation after each step, which can be written, using equation (35) as:

$$J_2\Delta\theta = J_2\left(I - J_2^{(I-H)\oplus}J2\right)He_1 + J_2J_2^{(I-H)\oplus}e_2 = J_2\Delta\theta^{*1} + J_2\Delta\theta^{*2}, \tag{36}$$

which can be splitted into:

$$F_A = J_2\Delta\theta^{*1} = J_2\left(I - J_2^{(I-H)\oplus}J2\right)He_1$$
$$F_B = J_2\Delta\theta^{*2} = J_2J_2^{(I-H)\oplus}e_2.$$

Now, having calculated $J_2^{(I-H)\oplus}$, we can use its singular value decomposition, keeping in mind that the result of this decomposition has to be expressed knowing $J_2^{(I-H)\oplus}$ is an inverse

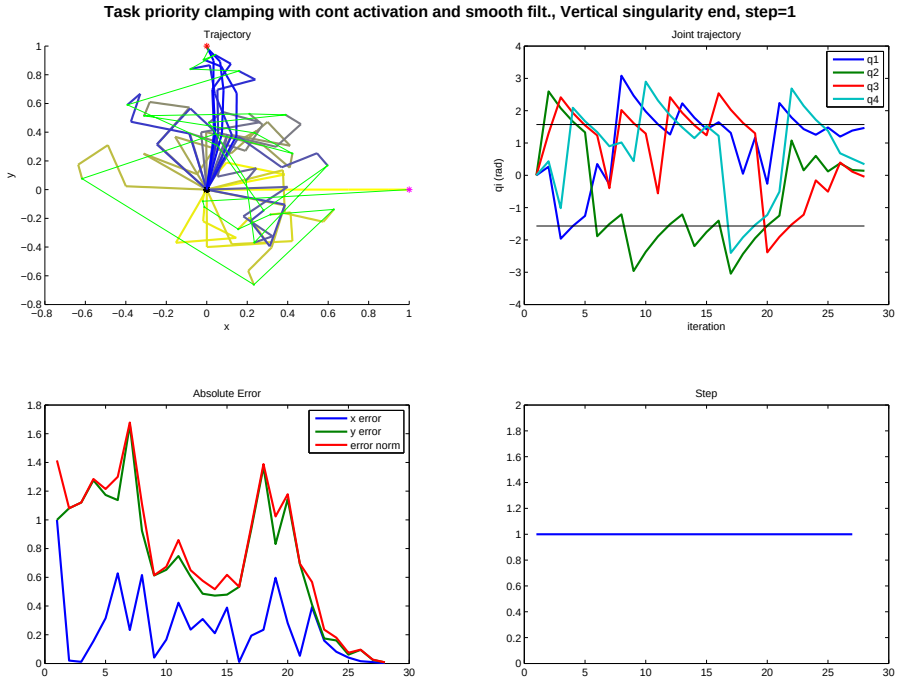

Figure 41: smoothly-filtered Jacobian continuous clamping

of J_2, thus

$$J_2^{(I-H)\oplus} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^T = \sum_{i=1}^{n} \hat{\sigma}^{-1}\hat{v}_i\hat{u}_i^T.$$

And by analogy to Section 5.3.4 assuming as in the selectively damped least squares algorithm, $e_2 = u_s$, with $s = 1..n$, we have $(u_k^T \cdot e_2) = (u_k^T \cdot u_s) = 1$ for $k = s$ and 0 otherwise, so we have

$$J^{(I-H)\oplus}e_2 = \sum_{i=1}^{n} \hat{\sigma}_i^{-1}\hat{v}_i\hat{u}_i^T,$$

and

$$J^{(I-H)\oplus}u_s = \hat{\sigma}_s^{-1}\hat{v}_s e_2,$$

which has an effect on each joint of:

$$\Delta\theta_j^s = \hat{\sigma}_s^{-1} J_2^j \hat{v}_{j,s}.$$

where $v_{j,s}$ is the jth position on the sth column of matrix V, and J_2^j is the jth column of matrix J_2.

Therefore, adding the norms for all joints as with M_i in equation (19), we have:

$$M_s = \hat{\sigma}_s^{-1} \sum_{j=1}^{m} |\hat{v}_{j,s}| \|J_2^j\|,$$

This M_s is a bound of the position change gain in the task space generated by the error-depending part of the algorithm, for each component of the error, and so with it we can set, for each $s = 1..n$, the maximum joints change:

$$\gamma_s = min(1, 1/M_s)\gamma_{max}$$

And now, proceed exactly as in Section 5.3.4.

Joints change for each error component (m-dimensional vector):

$$w_s = \hat{\sigma}_s^{-1}\hat{v}_s(\hat{u}_s^T \cdot e_2)$$

Damping these joints change:

$$\Delta q_s = \begin{cases} 1 & \text{if } \|w_s\| < \gamma_s \\ \frac{ws}{\|ws\|}\gamma_s & \text{if } \|w_s\| \geq \gamma_s \end{cases} \tag{37}$$

And then, we have to add the non error-dependent part of the algorithm to the sum of each component : $\Delta\hat{\theta} = (I - J_2^{(I-H)\oplus}J_2)He_1 + \sum_s q_s$

To finally damp the total step value:

$$\Delta\theta = \begin{cases} 1 & \text{if } \|\Delta\hat{\theta}\| < \gamma_{max} \\ \frac{\Delta\hat{\theta}}{\|\Delta\hat{\theta}\|}\gamma_{max} & \text{if } \|\Delta\hat{\theta}\| \geq \gamma_{max} \end{cases} \tag{38}$$

Now we will show the improvements of this method (with a previous filtering of the Jacobian as proposed in Section 5.4.1) with the same examples that we have been showing.

- Vertical singular end position. Now the computation time has been a bit longer, 0.1351s, but still affordable. As we can see in Figure 42, the method converges fastly and the error decreases at each step.
- Joint limit forcing. Now we can see in Figure 43, that the method arrives at its joint limits and begins with some chattering, as the goal cannot be achieved. Nevertheless, its behaviour is now much better than in the non selectively damped method, shown in Figure 33.
- Horizontal end position. Now, the algorithm solves the initial singularity, and converges fastly in 0.1654s with smooth steps and, looking at the joint trajectory, we can see that when joint 3 is approaching its minimum value of $\pi/2$, its tendency is damped so as to respect this limit.

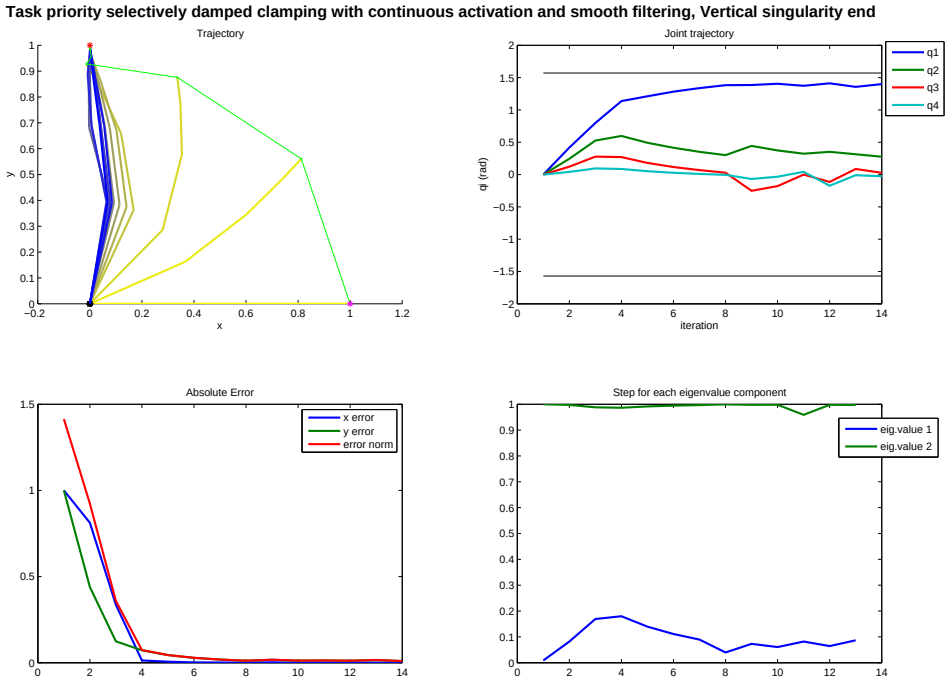

Figure 42: Proposed method for selectively damping the task-priority with continuous activation method. Bottom right graph plots the values of γ_s, for $s = 1, 2$

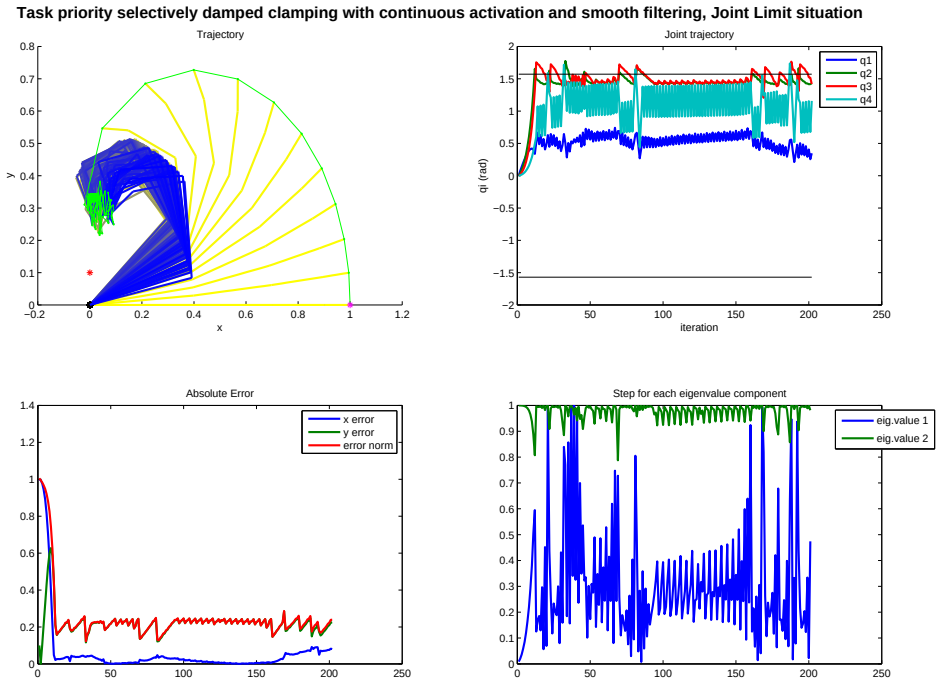

Figure 43: Proposed method for selectively damping the task-priority with continuous activation method

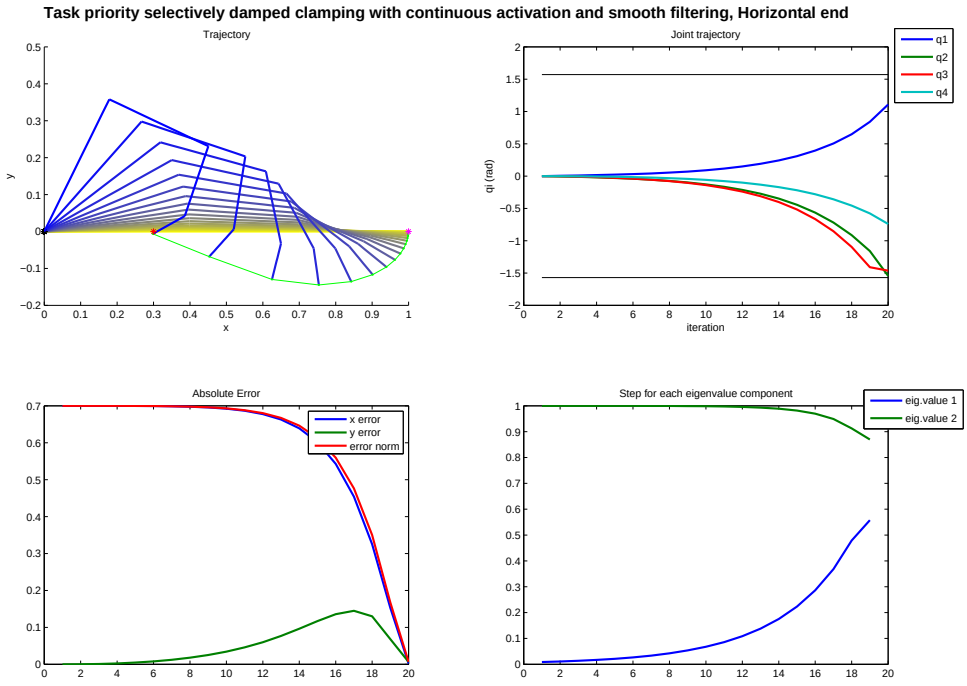

Figure 44: Proposed method for selectively damping the task-priority with continuous activation method

6 Experimental Results

Although the inverse kinematic methods have been described using a 4R planar robot for illustration purposes position control, a more complex experimentation with more practical robots must be done to assess their effectiveness. To that purpose, we have taken all the methods exposed, and used them to calculate the solution of a set of initial joint positions (the same set for all the methods) and final goals. We have compared the number of solutions found, the number of solutions found respecting the joint limits, and the average computational time for each one, for those whose solution is found, and also for those solutions respecting joint limits.

We have generally used a step of $\alpha = 1$ and random initial joint states, but also a singular initial joint state. The comparison is done on two robots: the 4R robot used before and the *Barrett's WAM* arm presented in section 4.4.3.

6.1 4R planar robot

To begin, we will analise the results for the 4R planar robot we have been using. To do so, first we have used 1000 different goal positions (all of them reachable) and checked whether the solution was found or not, with most of the presented algorithms, the robot always starting in the same joint position (all joints zero). We have set a maximum of 1000 iterations, and the results can be seen in Table 1, divided into two parts: the methods considering joint limits (bottom) and the methods which don't (top).

Method	% sol.	% sol. resp. limits	avg. time (ms)	avg. time sol. (ms)	resp. limits (ms)
J. Pseudoinverse	100.0	54.8	11	11	10
J. Transpose	100.0	97.7	27	27	27
J. Damping	100.0	55.0	11	11	10
J. Filtering	100.0	61.0	17	17	17
Selective Damping (SD.)	100.0	99.1	18	18	18
Gradient Projection	100.0	33.1	48	48	23
J. Smooth Filtering (SF)	100.0	36.9	12	12	13
TP. Clamping	93.7	89.1	340	233	212
J. Weighting	99.7	53.2	24	24	26
J. Clamping	95.0	69.5	107	20	14
Cont. TP Clamp.	98.7	92.9	98	84	76
Cont. TP Clamp & SF.	98,7	93.3	149	113	99
Cont. TP. Clamp, SF. & SD	99.8	99.1	52	52	51

Table 1: 4R robot results with all joints set at 0 at start position. TP stands for *Task Priority*. All methods using a step of $\alpha = 1$, except for those based on selective damping, using a different step on each eigenvector and the Jacobian transpose, which uses step in equation (17). In bold are the proposed methods

In this table we can observe the following:

- All the methods not considering joint limits converge.
  We must point out that, the proposed method to smoothly filter the Jacobian eigenvalues is faster than the Jacobian transpose and the Jacobian filtering algorithms, and almost as fast as the pseudoinverse and the Jacobian damping algorithm. This means that the proposed filtering not only converges, but also does not imply a loss of speed.

The methods respecting joint limits do not always converge due to the algorithmic singularities we have commented before. That is, in some points, the term pushing away from joint limits equals the term pushing to the goal.

- The Jacobian Transpose and the selective damping methods respect joint limits.
  This is due to the initial configuration, which is in the middle of the joint limit intervals. These methods do not guarantee the limits to be respected, but the way the step is calculated means the goal position is hardly ever surpassed.

- The Jacobian Weighting and the Jacobian Clamping methods are not effective in avoiding joint limits, as they don't reach 70% of the solutions respecting them.

- The methods supposedly respecting joint limits do not always achieve such a task.
  In these cases, what has happened is that in the step achieving the goal, one joint limit has been surpassed. On the following step, this joint limit would be inmediately corrected, but the algorithm has stopped as the ending criterion is only the position error under a tolerable norm. Nevertheless, as the step is more controlled with our proposed algorithm, the proportion of cases with solution not respecting joint limits is smaller.

- The proposed Continuous Clamping combined with the Selective Damping achieves almost all solutions, with a quite acceptable speed.

Now we have done the same calculations, but using a random joint configuration to start the algorithm, as can be seen in Table 2:

Method	% sol.	% sol. resp. limits	avg. time (ms)	avg. time sol. (ms)	avg. time resp. limits (ms)
J. Pseudoinverse	100.0	49.5	9	9	8
J. Transpose	100.0	66.3	53	53	60
J. Damping	100.0	49.7	9	9	8
J. Filtering	100.0	56.3	12	12	12
Selective Damping (SD.)	100.0	65.2	12	12	11
Gradient Projection	99.9	22.7	53	53	60
J. Smooth Filtering (SF)	100.0	50.2	9	9	8
TP Clamping	94.7	90.8	342	266	250
J. Weighting	98.8	48.0	42	28	39
J. Clamping	94.5	63.6	106	17	13
Cont. TP Clamp.	98.4	93.1	107	72	66
Cont. TP Clamp & SF.	98.6	93.0	127	81	77
Cont. TP Clamp, SF. & SD	93.3	90.0	30	51	47

Table 2: 4R planar manipulator. The initial position is set in a random (valid) joint configuration.

Now from this calculations we can extract some more details:

- The Jacobian Transpose and Selective Damping methods no longer respect joint limits, confirming our hypothesis that they achieved them because of the centered starting position.

- The performance of the proposed method of Continuous Clamping combined with the Selective Damping drops.
  Apparently, this result might be surprising. Nevertheless, if we take an example of these situations, we easily see what happens:
  In this case, we have an initial joint configuration of $\theta^0 = \begin{pmatrix} 1.1211 \\ 1.0413 \\ -0.4748 \\ 1.4810 \end{pmatrix}$ rad, which sets an initial position of $x^0 = \begin{pmatrix} -0.1167 \\ 0.8052 \end{pmatrix}$m, and a goal position at $x^G = \begin{pmatrix} 0.2430 \\ -0.8709 \end{pmatrix}$m.
  As we can see in Figure 45, both positions are almost opposite with respect to the origin. This means that the algorithm can approach the goal in a clockwise or counterclockwise direction. If it started clockwise, the goal would have been easily achieved. But its not the case, and the robot moves counterclockwise until it is stopped by the joint limit repulsive effect. As the algorithm's step is very smooth, it cannot jump to another configuration close to the goal. The solution is indeed found by the Continuous Task Clamping with Smooth Filtering, as with a step of $\alpha = 1$, as seen in Figure 46. There we can clearly see that this algorithm has found the solutions thanks to its big, almost random-like steps.

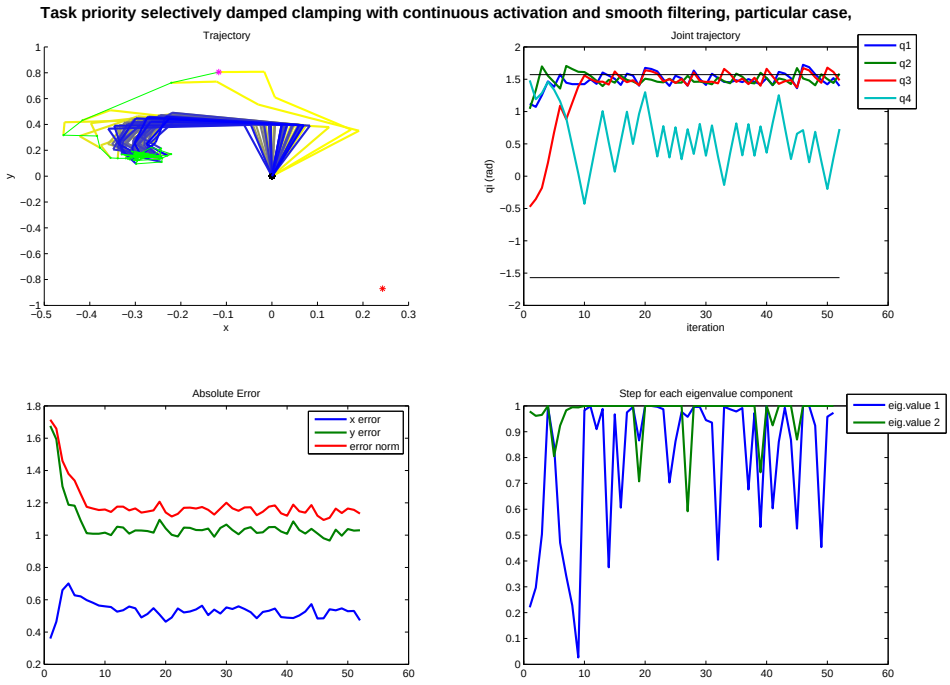

Figure 45: Proposed method getting stuck due to its continuity.

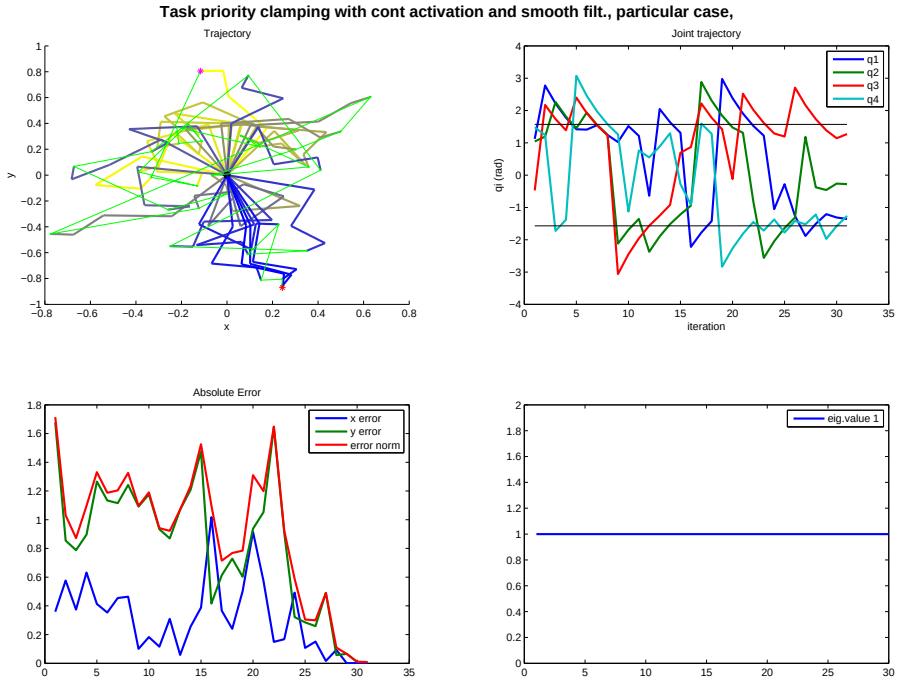

Figure 46: Chaotic steps can lead to goal.

6.2 Barret's WAM arm

For this robot studied in Section 4.4.3, the methods exposed for the 4R planar robot can be used, using the whole Jacobian matrix and error calculation as in Section 3.1. We will also compare to the analytical method exposed in section 4.4.3.

Performing the same analysis of the WAM arm manipulator, we have:

Method	% sol.	% sol. resp. limits	avg. time (ms)	avg. time sol. (ms)	avg. time resp. limits (ms)
J. Pseudoinverse	100.0	8.1	48	48	22
J. Transpose	60.6	34.9	582	439	432
J. Damping	100.0	9.3	44	44	22
J. Filtering	4.7	0	548	43	-
Selective Damping (SD.)	99.2	61.0	150	143	137
Gradient Projection	100.0	6.1	58	58	22
J. Smooth Filtering (SF)	100.0	0.2	42	42	17
TP. Clamping	0.3	0.3	611	24	24
J. Weighting	100.0	7.4	54	54	22
J. Clamping	43.4	14.5	537	113	86
Cont. TP Clamp.	29.5	29.4	4732	2714	2717
Cont. TP Clamp & SF.	29.9	29.7	4791	2832	2843
Cont. TP. Clamp, SF. & SD	62.1	62.0	3492	1742	1743
Analytical θ_3	100.0	100.0	324	324	324
Analytical optimized	100.0	100.0	316	316	316

Table 3: WAM arm results with joints set at 0 at start position. TP stands for *Task Priority*. All methods are using a step of $\alpha = 1$, except for those based on selective damping and the Jacobian transposed, which uses step in equation (17). Now two analytical methods have been added, Analytical θ_3 corresponds to the one that can be found in [11] and Analytical Optimized is the analytical method with the optimization explained in Section 4.4.3

We can observe the following:

- The Jacobian Transpose method's performance drops, being unable to find a solution, without concern on joint limits, in almost 40% of cases. This is due to what we commented in Figure 17, there is a chattering that slows the algorithm and, the more complicated the robot is, the more iterations will be needed to achieve the goal. We can also see the Jacobian Filtering method finds almost none solutions.
- The Smooth Filtering algorithm not only finds all solutions, but is also the fastest one of the joint limit unconcerned algorithms, confirming what we saw with the 4R robot.
- The Jacobian Weighting and Joint Clamping algorithms show no better performance than for the 4R robot, so these are not recommended. We also have to point out that the task-priority joint clamping, without the continuous w.r.t. activation matrix pseudoinverse, is unable to find solutions in this complex robot, so it is also discarted.
- The Selectively Damped method respects, as in the 4R robot, the joint limits, despite not taking them into account. As we have commented, this is due to the initial position and the step of the method, which has no *overshots* in its performance.
- The Continuous activation methods are slow. This is because of their computational complexity. Nevertheless, we see that adding a smooth filtering to the algorithm in equa-

tion (34) slightly improves its performance, and adding a selective damping almost doubles the number of solutions found.

- The analytical method as exposed in Section 4.4.3 is the fastest, and the most efficient. This could be expected, as we had an an almost completely analytical solution for it.

Now if we use random starting positions, the results are:

Method	% sol.	% sol. resp. limits	avg. time (ms)	avg. time sol. (ms)	avg. time resp. limits (ms)
J. Pseudoinverse	100.0	6.3	45	45	20
J. Transpose	54.8	15.4	645	469	471
J. Damping	100.0	6.3	44	44	19
J. Filtering	100.0	7.5	39	39	20
Selective Damping (SD.)	98.6	30.2	158	148	125
Gradient Projection	100.0	14	57	57	19
J. Smooth Filtering (SF)	100.0	7.3	38	38	18
TP Clamping	0.7	0.7	810	23	23
J. Weighting	100.0	5.1	53	53	20
J. Clamping	56.4	21.5	460	119	88
Cont. TP Clamp.	35.8	35.8	4381	2395	2395
Cont. TP Clamp & SF.	36.3	36.2	4365	2322	2322
Cont. TP Clamp, SF. & SD	46.9	46.9	4516	1884	1884
Analytical θ_3	100.0	100.0	326	326	326
Analytical optimized	100.0	100.0	307	307	307

Table 4: WAM arm manipulator. Now the initial position is set in a random (valid) joint configuration.

Where we can extract the same conclusions. Now our proposed method's performance does not reach 50% of solutions, but it is still the best of the used control-based algorithms.

The low convergence rate of this last experiment is due to a large initial error in the first steps. In some cases, when having such a large initial error, these algorithms perform random steps until they reach a position in which the error is small enough to converge. Nevertheless, these algorithms can be helped by performing some path planning, in the sense of calculating a trajectory to reach a goal position, using intermediate points so the error in each part of the algorithm will be smaller, these methods will more easily converge, acting now as local solution finding algorithms.

To show how these methods behave in this sense, we have designed a trajectory with the WAM arm, by the equations

$$x_0 = f\begin{pmatrix} 0 \\ 0 \\ 0 \\ 2.2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$x_f = x_0 + \begin{pmatrix} -0.05t \\ 0.2tcos(4\pi t) \\ 0.1sin(4\pi t) \end{pmatrix},$$

where f is the forward kinematics mapping.

With t going from 0 to 1 divided in 50 equidistant steps. The result for some of the methods commented are shown in Figures 48 to 54, showing separately the evolution of each joint, and the error in Figure 47. In these plots, we can see blue vertical stripes, which represent the possible values of each joint at each position of the trajectory. This is calculated by solving, for all possible fixed angles, the equations in Section 4.4.3.

In this case, the methods not respecting joint limits have all converged to zero error, except the Jacobian Transpose. To show cleaner results, we will plot only the joint limit respecting methods: Jacobian Clamping, Jacobian Weighting, Continuous Task Priority Clamping, the Continuous Task Priority Clamping with the Smooth Filtering of the Jacobian, and the proposed method in Section 5.4.2. We do not plot the Task Priority Clamping because it does not converge.

With these pictures, we can extract the following:

- The Jacobian Weighting method, although converging, is unable to respect joint limits and performs large jumps.
- The Jacobian Clamping method converges, but making big leaps in its performance and, for instance, keeping joint 7 blocked most of the time. These leaps are due to changes in the *assembly modes* of the solutions. As we said in Section 4.4.3, the robot has different assembly modes (inner elbow, outer elbow, shoulder orientation, wrist orientation, etc.). When a trajectory is being done in one assembly mode, and it reaches a point in which this assembly mode can no longer follow the trajectory, there is a bigger change in the joints position, corresponding to a change of the assembly mode being tracked. This leads to fast and dangerous movements of the robot, which must be avoided.
- The Continuous Task priority algorithm is unable to converge without our proposed filtering of the Jacobian matrix. This highlights how useful can be to filter the Jacobian, so as to avoid unstabilities due to its loss of rank.
- Our proposed method converges, respecting joint limits all the time, and makes the smallest leaps among the described methods, thus being the smoothest algorithm. Nevertheless, we can also see some *holes* in the blue solution stripes. These holes correspond to intervals for the articulation in which no assembly mode can find any solution. In some cases, when these holes' boundaries have values close to a joint limit, the solution found stays at this boundary, due to the repulsion effect of the joint-centering task.

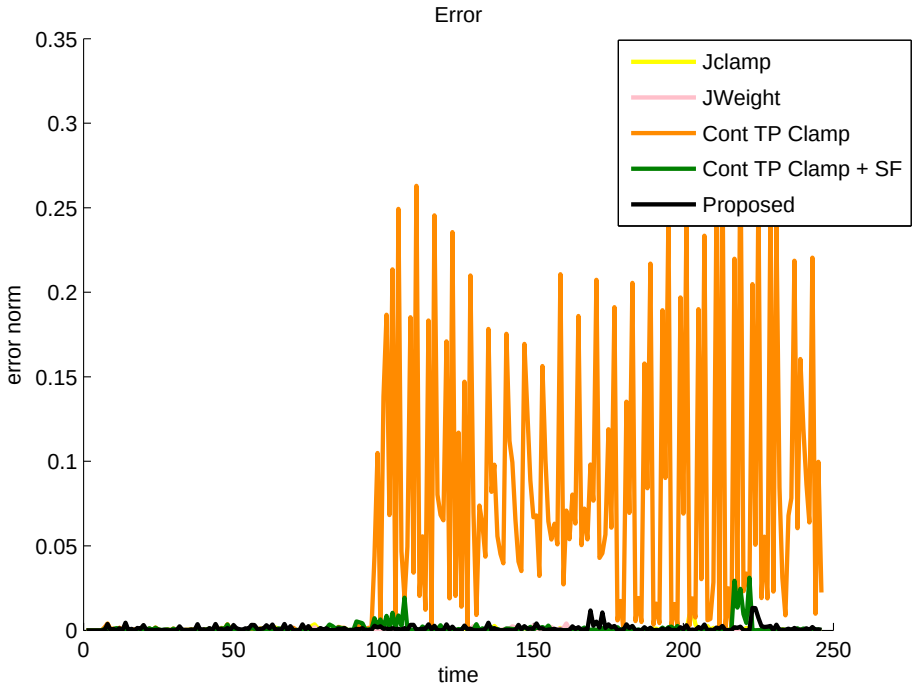

Figure 47: As we can see, all methods converge except the Continuous (with respect to activation matrix) Task Priority Method without the smooth filtering of the Jacobian matrix

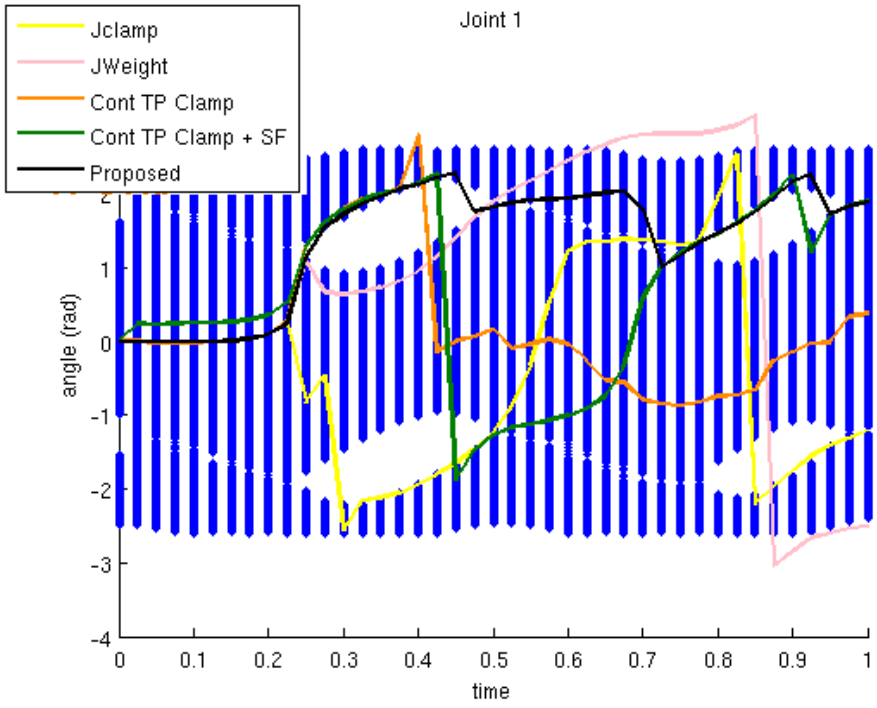

Figure 48: Joint 1 evolution along the trajectory. As we can see, our proposed methods performs the smallests leaps.

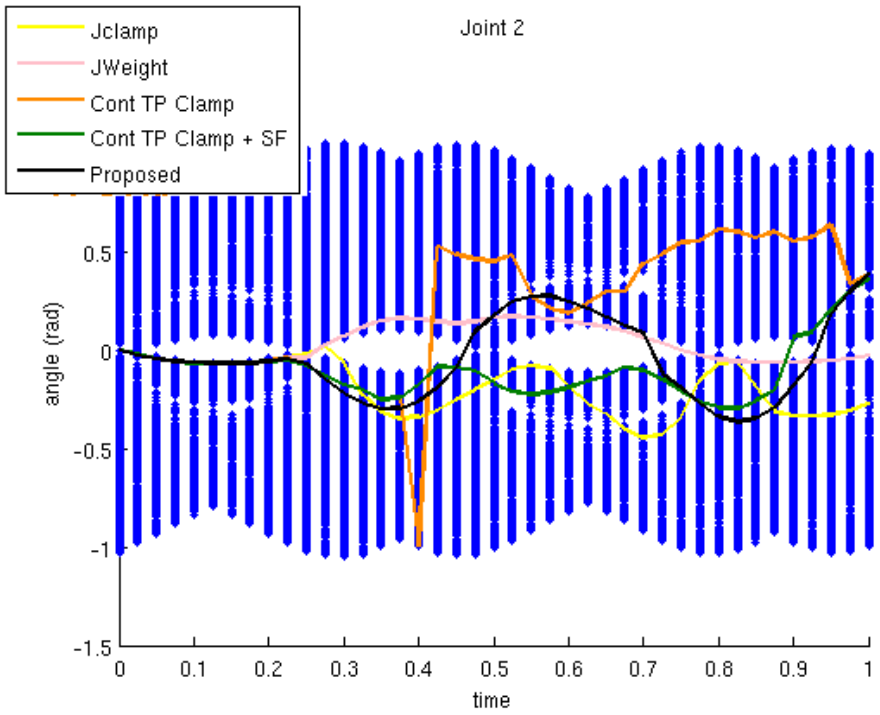

Figure 49: Joint 2 evolution along the trajectory.

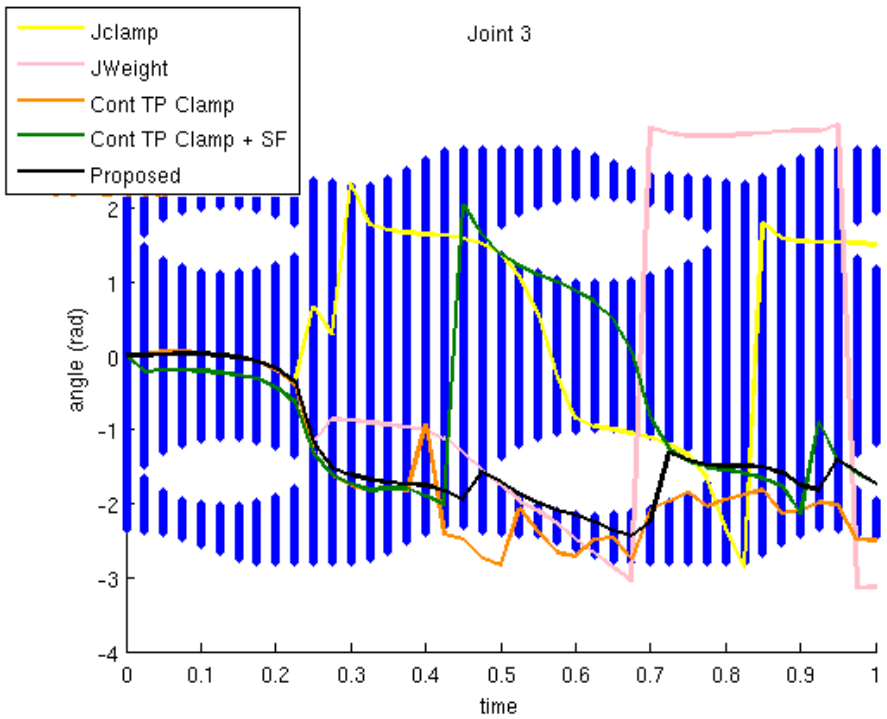

Figure 50: Joint 3 evolution along the trajectory.

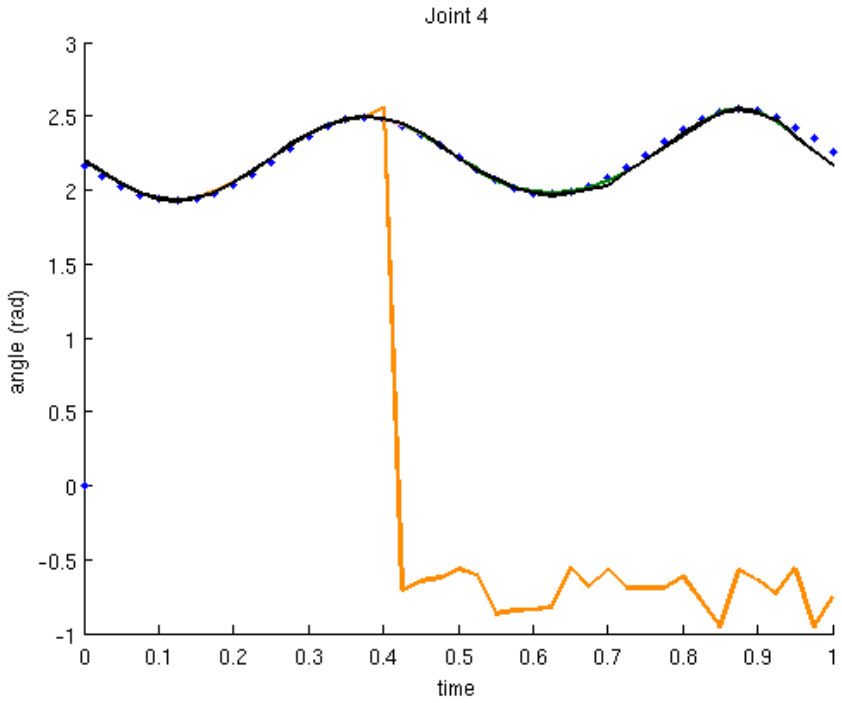

Figure 51: Joint 4 evolution along the trajectory.

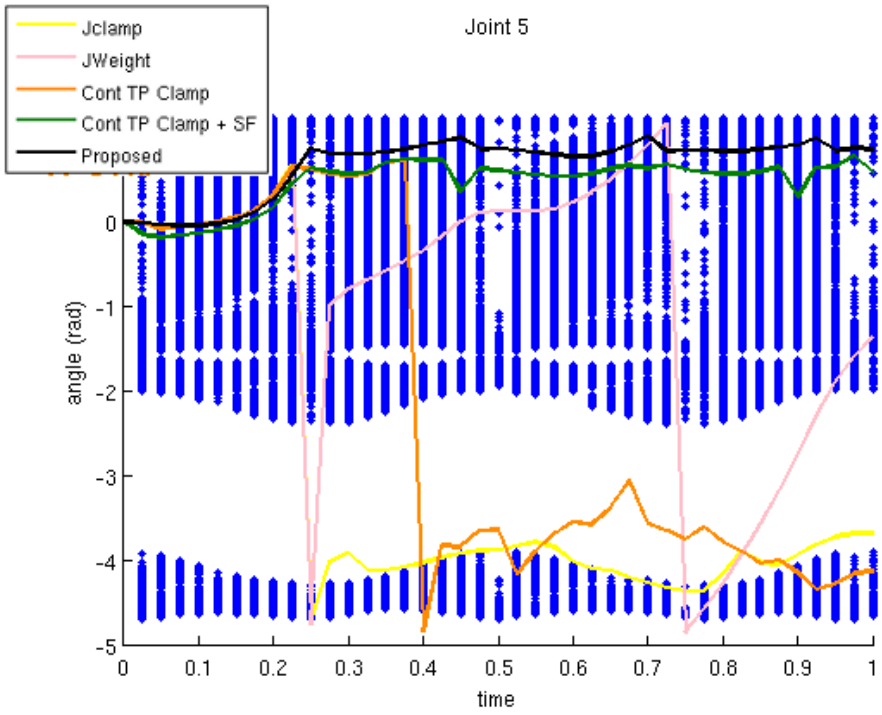

Figure 52: Joint 5 evolution along the trajectory.

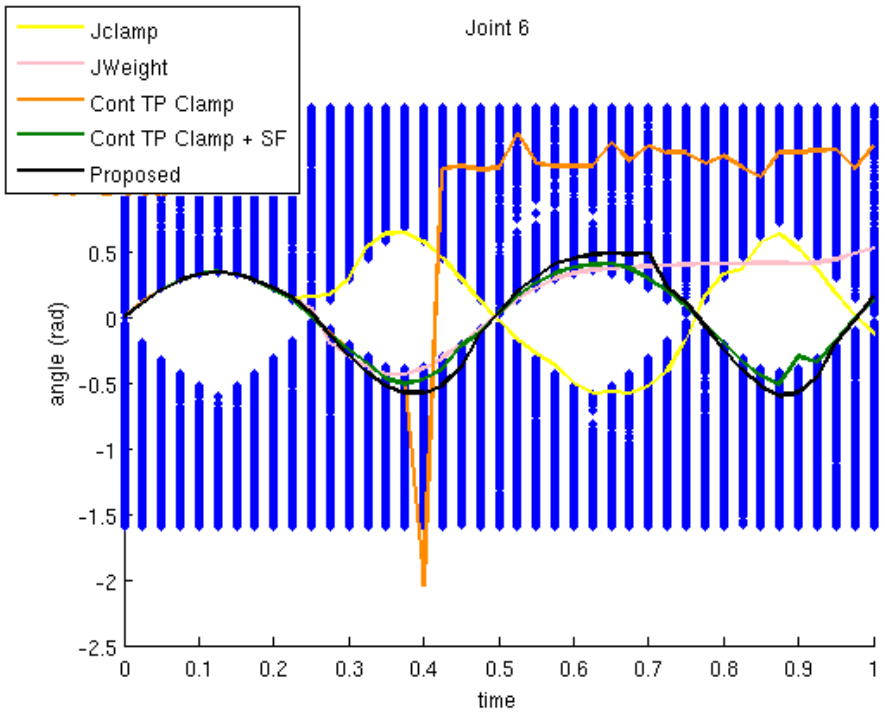

Figure 53: Joint 6 evolution along the trajectory.

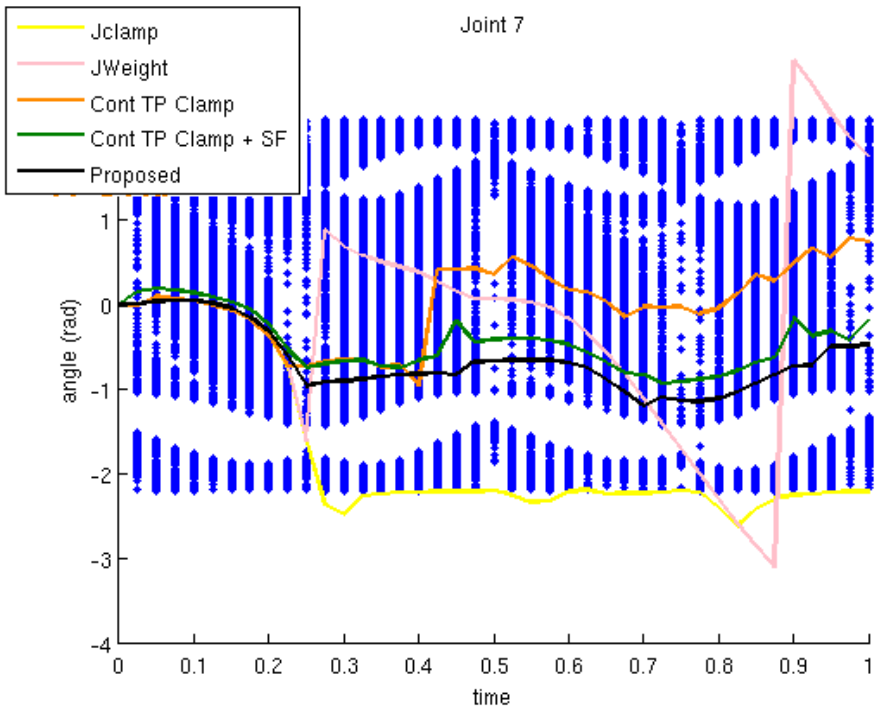

Figure 54: Joint 7 evolution along the trajectory.

7 Conclusions and Future Work

Along this work, we have exposed the most popular of the inverse kinematic control-based algorithms in a survey-like way, clearly pointing their pros and contras. We have also proposed two ways of improving these methods:

- Analytical methods are usually faster. With the example of the WAM arm, we have clearly seen that analytical methods are always a good solution when available. In this case, for the WAM arm, we could find such a solution, but for the case of the 4R manipulator, were it is harder to determine it, or for the laparoscopic robot commented, alternative methods such as control-based ones are a good option.
- Efficiently filtering the Jacobian matrix of a manipulator.
  As exposed in Section 5.4.1, we proved theoretically and in practice that our proposal improves the existing methods to numerically filter or damp the Jacobian pseudoinverse of a matrix. We have also seen that this does not mean an additional computational cost to the algorithm. With this filtering, the Jacobian matrix can be assumed to always be full ranked, without generating much additional error on the algorithms, thus the pseudoinverse operator does not have discontinuities due to a rank change in the Jacobian matrix. This can be used in all control-based methods so as to improve their performance.
- A smooth inverse kinematic method.
  Smooth not only in its dependency on the rank of the Jacobian matrix, but also in its activation matrices to avoid joint limits, thanks to the work done at [14]. Also, its step, different for each eigenvector of the task, using what was proposed in [2], avoids large gains on each iteration. This method combines the advantages of other existing procedures to approach the target position **smoothly**, eliminating any *random-like* behaviour, with the drawback that being smooth means it cannot find the solution *by chance*, as those with a large step can do when the initial error is too large. This proposed algorithm might perform even better in relation to the other algorithms seen when using it as a position online control algorithm, or a local optimizing algorithm as it is robust with respect to the error, respects joint limits, and calculates its step at each iteration. More secondary tasks can also be added with a task priority structure, to improve the obtained position.

As stated in [7], the convergence of control-based methods strongly depends on their step α and their initial error e_0. If α is set as a constant and the initial error is large, then these methods perform oriented jumps until they reach a position in which the error is small enough to make the method converge with the given α. In fact, this behaviour could be linked with pseudo-random planning algorithms such as the *Rapidly-exploring Random Tree* algorithm. When used as local planners for complex robots, although their convergence improves, it cannot be assured without considering the iteration step α, and these methods easily jump from one assembly mode to another, causing dangerous moves of the manipulator if controlled online. Our proposed method reduces these drawbacks.

From here, our future work on it could be oriented in the following directions:

- Add other optimization tasks on the proposed algorithm, such a manipulability optimization.
- Reduce computational cost, possibly by tuning parameters and improving the algorithms.
- Avoid the algorithm to take *bad* decisions as in Figure 45. This issue might lead to the path planning field.

References

[1] J. Baillieul. Kinematic programming alternatives for redundant manipulators. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation.*, volume 2, pages 722–728, mar 1985.

[2] Samuel R. Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools*, 10:37–49, 2004.

[3] Tan Fung Chan and R.V. Dubey. A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators. *IEEE Transactions on Robotics and Automation*, 11(2):286–292, apr 1995.

[4] S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *Robotics and Automation, IEEE Transactions on*, 13(3):398–410, jun 1997.

[5] S. Chiaverini, O. Egeland, and R.K. Kanestrom. Achieving user-defined accuracy with damped least-squares inverse kinematics. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, pages 672–677 vol.1, jun 1991.

[6] S. Chiaverini, B. Siciliano, and O. Egeland. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *Control Systems Technology, IEEE Transactions on*, 2(2):123–134, jun 1994.

[7] P. Falco and C. Natale. On the stability of closed-loop inverse kinematics algorithms for redundant robots. *Robotics, IEEE Transactions on*, PP(99):1–5, 2011.

[8] Z Jiang. A converse lyapunov theorem for discrete-time systems with disturbances. *Systems & Control Letters*, 45:49–58, 2002.

[9] C.A. Klein, C. Chu-Jenq, and S. Ahmed. A new formulation of the extended jacobian method and its use in mapping algorithmic singularities for kinematically redundant manipulators. *Robotics and Automation, IEEE Transactions on*, 11(1):50–55, feb 1995.

[10] Jihong Lee. A study on the manipulability measures for robot manipulators. In *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 3, pages 1458–1465 vol.3, sep 1997.

[11] T. Lozano-Perez and L. Kaelbling. Barrett wam/hand interface ros-pkg. http://code.google.com/p/lis-ros-pkg/.

[12] N. Mansard, O. Khatib, and A. Kheddar. A unified approach to integrate unilateral constraints in the stack of tasks. *Robotics, IEEE Transactions on*, 25(3):670–685, june 2009.

[13] N. Mansard, A. Remazeilles, and F. Chaumette. Continuity of varying-feature-set control laws. Technical report, IRISA, 2007. Downloadable version at ftp://ftp.irisa.fr/techreports/2007/PI-1864.pdf.

[14] N. Mansard, A. Remazeilles, and F. Chaumette. Continuity of varying-feature-set control laws. *Automatic Control, IEEE Transactions on*, 54(11):2493–2505, nov. 2009.

[15] Josep M. Porta, Lluís Ros, and Federico Thomas. Inverse kinematic solution of robot manipulators using interval analysis. 4th International Workshop on Computational Kinematics, 2005.

[16] R. S. Rao, A. Asaithambi, and S. K. Agrawal. Inverse kinematic solution of robot manipulators using interval analysis. *Journal of Mechanical Design*, 120(1):147–150, 1998.

[17] D. Raunhardt and R. Boulic. Progressive clamping. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4414–4419, april 2007.

[18] M. Shimizu, H. Kakuya, W.-K. Yoon, K. Kitagaki, and K. Kosuge. Analytical inverse kinematic computation for 7-dof redundant manipulators with joint limits and its application to redundancy resolution. *Robotics, IEEE Transactions on*, 24(5):1131–1142, oct. 2008.

[19] B. Siciliano and J.-J.E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, pages 1211–1216 vol.2, jun 1991.

[20] Bruno Siciliano, Lorenzo Sciavicco, Luifi Villani, and Giuseppe Oriolo. *Robotics. Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer, 1st edition, 2009.

[21] G.K. Singh and J. Claassens. An analytical solution for the inverse kinematics of a redundant 7dof manipulator with link offsets. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2976–2982, oct. 2010.

[22] Sreenivas Tejomurtula and Subhash Kak. Inverse kinematics in robotics using neural networks. *Information Sciences*, 116(2-4):147–164, 1999.

[23] D. E. Whitney. Resolved motion rate control of manipulators and human prostheses. In *IEEE Transactions on Man-Machine Systems, 10*, pages 47–53, 1969.

[24] W. A. Wolovich and H. Elliott. A computational technique for inverse kinematics. In *Decision and Control, 1984. The 23rd IEEE Conference on*, volume 23, pages 1359–1363, dec. 1984.

[25] T. Yoshikawa. Dynamic manipulability of robot manipulators. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 1033–1038, mar 1985.

IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.