Semester Project

Implementation of the singularity free inverse kinematic solution for redundant robots

Benjamin Ben Hattar

benjamin.benhattar@epfl.ch

Supervised by
Prof. Aude Billard
Seyed Sina Mirrazavi Salehian
Nadia Figueroa

January 18, 2016

Abstract

This project is about the implementation of a new algorithm to solve the inverse kinematics problem. Unlike many existing algorithms, it is not based on the pseudoinverse of the Jacobian but on Quadratic Programming so it is more robust to singularities. The algorithm implemented solves the inverse kinematics problem on the velocity level and minimizes the kinetic energy.

The solver that we have implemented in C++ has been tested on simulator and it efficiently minimizes the kinetic energy. The precision was however lower than the one obtained with the damped least square based solver currently implemented in the Robot ToolKit. The implementation on a real KUKA LWR was successful as the real motion was very similar to the simulated one, this shows that the algorithm is robust to noise.

Contents

Figure 1: *KUKA lightweight robot (LWR).*

1 Introduction

The control of a robotic arm, like the KUKA LWR presented on figure 1, is done by moving its joints. It is however difficult to plan a motion of the robot's end-effector by setting the motion of the joints directly. It is hence useful to have an efficient method, called inverse kinematics, to translate the desired end-effector motion into a joint motion.

1.1 The inverse kinematics problem

The inverse kinematics problem on the velocity level can be formulated as follows: From the desired velocity of the end-effector of a robot, how can we find the velocity of its joints?

In the case of a redundant robot, the answer to this question is far from trivial. Indeed, as shown on the figure 2, for a desired position of the end-effector there are many different configurations of the joints. The redundancy makes the inverse kinematics problem more complex but it allows optimization through null-space motion; the null-space motion being the internal motion of the joints that has no effect on the end-effector's position.

In this report, we will consider a robot which has n degrees of freedom and which end-effector has m degrees of freedom with $n > m$, the redundancy condition.

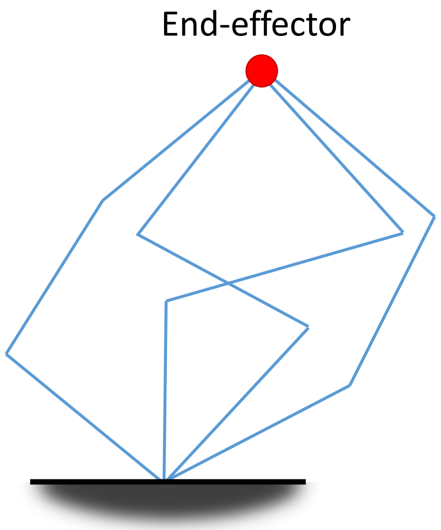


Figure 2: *In a redundant robot, an infinite number of configurations of the joints can lead to the same end-effector position.*

The forward kinematics on the velocity level can be formulated as follows:

$$\dot{r} = J(\theta)\dot{\theta} \tag{1}$$

$r \in \mathbb{R}^m$ being the position of the end-effector, $\theta \in \mathbb{R}^n$ the joints angles or more generally positions and $J \in \mathbb{R}^{m \times n}$ is the Jacobian. The Jacobian can be computed from the joints positions so it is a simple task to determine the end-effector velocity $\dot{r}$ from the joints velocities $\dot{\theta}$.

The inverse problem is not as simple. The Jacobian is not square so it cannot inverted simply.

1.1.1 The pseudo-inverse of the Jacobian method

A natural way to tackle the inverse kinematics problem is to take the pseudo-inverse of the Jacobian $J^\dagger$. The joints velocities are then found as:

$$\dot{\theta} = J^\dagger \dot{r} \tag{2}$$

This method works well but if the Jacobian is rank deficient, or singular, this method doesn't give the desired velocity.

1.1.2 The damped least square method

One solution to avoid the singularities is the damped least square method:

$$\dot{\theta} = \begin{cases} J^{\dagger}\dot{r} & \text{if } J \text{ is not singular} \\ (J+\lambda I)^{\dagger}\dot{r} & \text{if } J \text{ is near singular} \end{cases} \tag{3}$$

$I \in \mathbb{R}^{m\times n}$ is the $m \times n$ identity matrix, *i.e.* a rectangular matrix with 1 on the diagonal and 0 elsewhere. λ is a small positive value. It avoids the singularities but since $(J+\lambda I)^{\dagger}J \neq I$, the actual velocity of the end effector will not be the desired one.

1.1.3 The transposed of the Jacobian method

Another possibility is to take the transposed of the Jacobian instead of its pseudo-inverse:

$$\dot{\theta} = \alpha J^T \dot{r} \tag{4}$$

α being a positive number.

The transposed is not equal to the pseudo-inverse but it is possible to justify [2] that it works under some approximation if α is chosen well.

2 Quadratic-programming based minimum kinematic energy method

In order to both avoid the singularities and minimize the kinetic energy, a more recent approach called quadratic-programming (QP) can be used [1].

The problem is as follows:

$$\min \frac{\dot{\theta}^T H \dot{\theta}}{2} \tag{5}$$

$$s.t. \quad \dot{r} = J\dot{\theta} \tag{6}$$

$$\theta^- \leq \theta \leq \theta^+ \tag{7}$$

$$\dot{\theta}^- \leq \dot{\theta} \leq \dot{\theta}^+ \tag{8}$$

The matrix $H \in \mathbb{R}^{n\times n}$ in equation 5 is the inertia matrix. θ^- and θ^+ are the joints position limits. $\dot{\theta}^-$ and $\dot{\theta}^+$ are the joints velocity limits.

From a given desired end-effector velocity, the algorithm will find the required joints velocities (equation 6) such that the kinetic energy is minimized (equation 5). There are inequality limits both on the joints position (equation 7) and on their velocities (equation 8).

When minimizing the kinetic energy, we actually minimize the motion of the joints that are further away from the end-effector, *i.e.* those that have to move more mass.

In order to deal only with velocities in the resolution of the problem, the equations 7 and 8 are simplified:

$$\xi^{-} \leq \dot{\theta} \leq \xi^{+} \tag{9}$$

With:

$$\begin{aligned} \xi^{-} &:= \max\left(\mu_p(\theta^{-} - \theta(t)), \dot{\theta}^{-}\right) \\ \xi^{+} &:= \min\left(\mu_p(\theta^{+} - \theta(t)), \dot{\theta}^{+}\right) \end{aligned} \tag{10}$$

μ_p is a hyper-parameter of the algorithm, it tunes how the joints limits will be avoided in a smooth way. A smaller value of μ_p means that the joints limits will be avoided from further away to be reached.

This specific problem is a part of a more general class of problems called quadratic programming. This class of problem includes acceleration level solutions to minimize the torque of the joints or the acceleration norm. The formulation being very similar between quadratic programming problem it is possible to use the same solver.

2.1 An efficient solver

In order to solve the problem in an efficient way, it has been proved[1] that it can be solved in the following way.

First a decision vector u and its limits are are defined:

$$u := \begin{pmatrix} \dot{\theta} \\ \dot{r} \end{pmatrix} \in \mathbb{R}^{n+m} \tag{11}$$

$$u^{-} := \begin{pmatrix} \xi^{-} \\ -\dot{r}^{+} \end{pmatrix}, u^{+} := \begin{pmatrix} \xi^{+} \\ \dot{r}^{+} \end{pmatrix} \tag{12}$$

$\dot{r}$ being the desired velocity and $\dot{\theta}$ the current velocity of the joints.

As there is no reason to limit the end effector velocity, $\dot{r}$ can be set as $+\infty$ or numerically as a very large number.

The matrix M and the vector q are needed in the computations:

$$M = \begin{pmatrix} H & -J^T \\ J & 0 \end{pmatrix} \in \mathbb{R}^{n+m \times n+m} \tag{13}$$

$$q = \begin{pmatrix} 0 \\ -\dot{r} \end{pmatrix} \in \mathbb{R}^{n+m} \tag{14}$$

The Jacobian $J = J(\theta)$ is needed in equation 13 so the current position of the joints has to be given to the solver.

The projection operator $P_\Omega(\cdot)$ is defined as:

$$P_\Omega(u) = [P_\Omega(u_1), ..., P_\Omega(u_{n+m})] \tag{15}$$

$$P_\Omega(u_i) = \begin{cases} u_i^- & \text{if} \quad u_i < u_i^- \\ u_i & \text{if} \quad u_i^- \leq u_i \leq u_i^+ \\ u_i^+ & \text{if} \quad u_i > u_i^+ \end{cases} \quad , \quad \forall i = 1, 2, ...n+m \tag{16}$$

The time-derivative of the decision vector u is then given by:

$$\dot{u} = \gamma(I + M^T)[P_\Omega(u - (Mu + q)) - u] \tag{17}$$

γ is a hyper parameter that scales the convergence

The new value of u after a time step Δt is now given by:

$$u(t + \Delta t) = u(t) + \int_t^{t+\Delta t} \dot{u}(t')dt' \tag{18}$$

The forward Euler integration scheme gives an approximation of the new value of u:

$$u(t + \Delta t) = u(t) + \Delta t \cdot \dot{u}(t) \tag{19}$$

The desired velocity of the joints $\dot{\theta}_D$ is then given by the first n components of u.

A block diagram of the algorithm is presented on the figure 3.

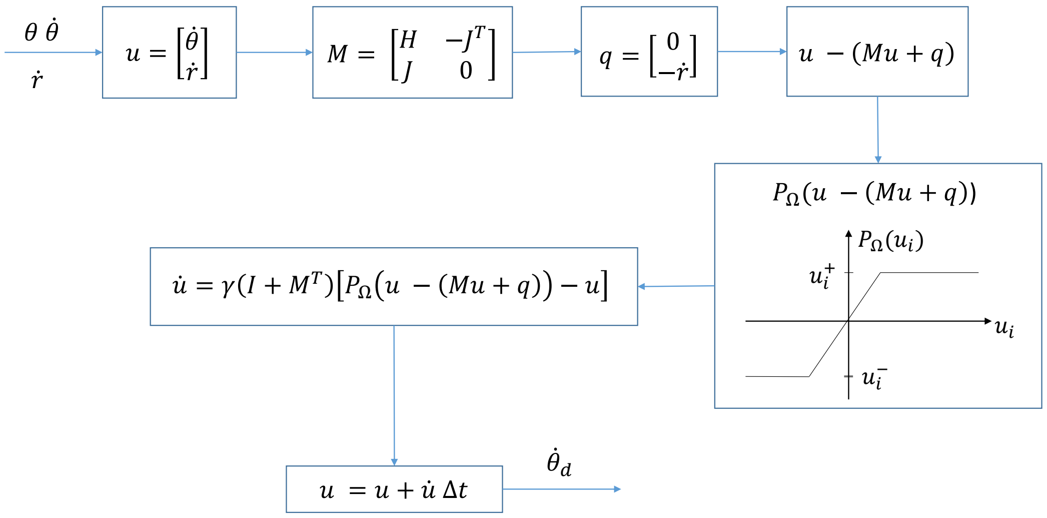
2.2 Analysis of the method

The method presented above has the advantage to require no pseudo-inversion of the Jacobian that are not only computationally costly but that can also be the source of algorithmic singularities.

The algorithm optimizes the kinetic energy which can be useful to diminish energy consumption of robots.

It is possible to minimize measures other than the kinematic energy with very similar procedures [5]. For example, by taking the identity matrix instead of H in equation 5 the algorithm will minimize the joints velocities total norm.

There is error arising from the numerical integration in equation 19. This error is of the order of the time-step Δt [3].



$\theta\ \dot\theta$
$\dot r$
$u = \begin{bmatrix}\dot\theta \\ \dot r\end{bmatrix}$
$M = \begin{bmatrix}H & -J^T \\ J & 0\end{bmatrix}$
$q = \begin{bmatrix}0 \\ -\dot r\end{bmatrix}$
$u - (Mu + q)$
$P_\Omega(u - (Mu + q))$
$P_\Omega(u_i)$
u_i^+
u_i
u_i^-
$\dot u = \gamma(I + M^T)[P_\Omega(u - (Mu + q)) - u]$
$u = u + \dot u\,\Delta t$
$\dot\theta_d$

Figure 3: *Block diagram of the quadratic programming solver for the minimum kinetic energy inverse kinematics problem.*

3 Acceleration level inverse kinematics

Given that quadratic programming is a general method for solving optimization problems, it can also be adapted to acceleration level inverse kinematics. On the acceleration level, a desired acceleration of the end effector $\ddot r$ is given to the solver and the desired joints accelerations $\ddot\theta$ are found [1].

A possible acceleration level inverse kinematics problem is the minimization of the acceleration norm:

$$\min \frac{\ddot\theta^T I \ddot\theta}{2} \tag{20}$$

$$s.t. J\ddot\theta = \ddot r_\alpha \tag{21}$$

$$\theta^- \leq \theta \leq \theta^+ \tag{22}$$

$$\dot\theta^- \leq \dot\theta \leq \dot\theta^+ \tag{23}$$

$$\ddot\theta^- \leq \ddot\theta \leq \ddot\theta^+ \tag{24}$$

Where I is the $n \times n$ identity matrix and $\ddot r_\alpha := \ddot r - \dot J(\theta)\dot\theta$.

Like for the velocity level resolution, it is possible to simplify the limits defined in equations 22, 23 and 24 into:

$$\xi^- \leq \ddot\theta \leq \xi^+ \tag{25}$$

With:

$$\begin{aligned} \xi^- &:= \max\left(\mu_p(\eta_p\theta^- - \theta(t)), \mu_v(\dot{\theta}^- - \dot{\theta}(t)), \ddot{\theta}^-\right) \\ \xi^+ &:= \min\left(\mu_p(\eta_p\theta^+ - \theta(t)), \mu_v(\dot{\theta}^+ - \dot{\theta}(t)), \ddot{\theta}^+\right) \end{aligned} \tag{26}$$

μ_p, η_p and μ_v are hyper parameters that tune how the constraints will be avoided.

The resolution is very similar to what has been done for the minimum kinematic energy in 2.1, the details can be found in Zhang's article [1].

4 Numerical simulation

4.1 Motion control

In order for the robot to follow a given desired trajectory given by parametric equations, the solver needs to receive an appropriate end effector velocity.

Let's call r_d the desired position. r is the actual position of the end effector.

If the end effector velocity $\dot{r}$ is simply set to the desired velocity:

$$\dot{r} = \dot{r}_d \tag{27}$$

The error on the trajectory will constantly accumulate over time.

To avoid that, the velocity should not only follow the desired velocity but also track the position.

$$\dot{r} = \dot{r}_d + g(r_d - r) \tag{28}$$

Where g is a positive gain parameter.

4.2 Parameters of the simulation

In order to test the quadratic based algorithm presented, a numerical simulation of a 7 degrees of freedom KUKA Light Weight Robot is performed. The desired motion of the end-effector is as follow:

$$\begin{cases} x = & A\sin(\sin^2(kt)) \\ y = & A\cos(\sin^2(kt)) \\ z = & C + D\sin(4kt) \end{cases} \tag{29}$$

A, C, D and k are constants and t is the time. For the following simulations $A = 50cm$, $C = 60cm$, $D = 20cm$ and $k = 0.4$.

The trajectory given by equation 29 is a sort of curved infinity shape and is represented on figure 6. The motion created by the parametric equation 29 is periodic with a period $\frac{\pi}{k}$.

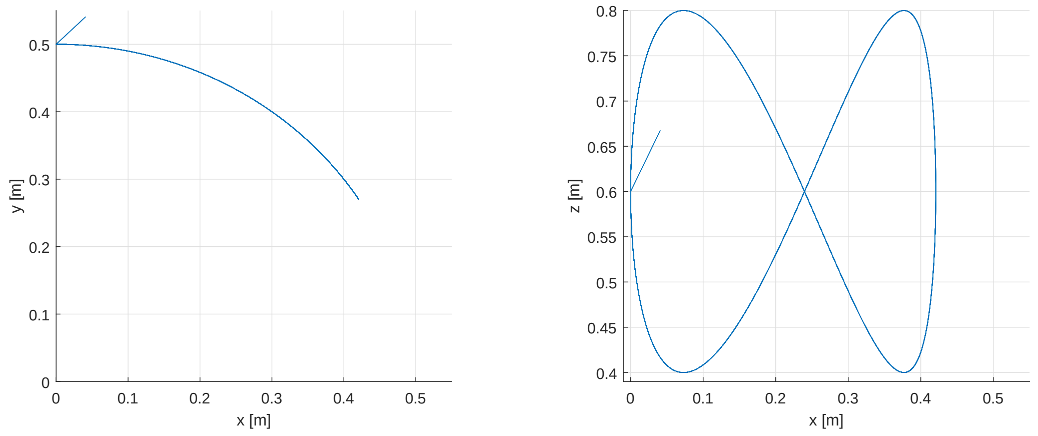


Figure 4: Projections of the desired trajectory on the XY and XZ planes.

The inertia matrix of the robot is a diagonal matrix with 1 for the four joints that are the closest to the base and 0.1 for the other ones. It means that the algorithm will try to avoid moving the joints near the base of the robot.

Before following the trajectory defined by equation 29, the end-effector motion is set to go to the desired starting position at $t = 0$ with a constant velocity from its initial position that is set to be close to the starting position.

Projections of the desired trajectory are represented on the figure 4.

The hyper-parameters of the solver are $\gamma = 50$ and $\mu_p = 25$. As mentioned before, finding good values for these hyper-parameters is not an easy task. For this report it has been done by simply tuning the parameters and observation of the result but an automatic grid search trying to optimize a performance measure based on the average error norm on the position and the stability would be better.

Since the inverse kinematics problem is solved at the velocity level, a velocity of the end-effector v has to be given to the solver. Let's call r_d the desired position and v_d its time-derivative. Setting $v = v_d$ would result in a drift of the position as the error would accumulate over time, this is why a term of positional tracking should be added:

$$v = v_d + g \cdot (r_d - r) \tag{30}$$

Where g is a gain parameter that will be set to $g = 0.2$. As the equation for the position is relatively simple, v_d can be found analytically with Mathematica. In the case of more complex trajectories, v_d can be found by numerical differentiation of r_d.

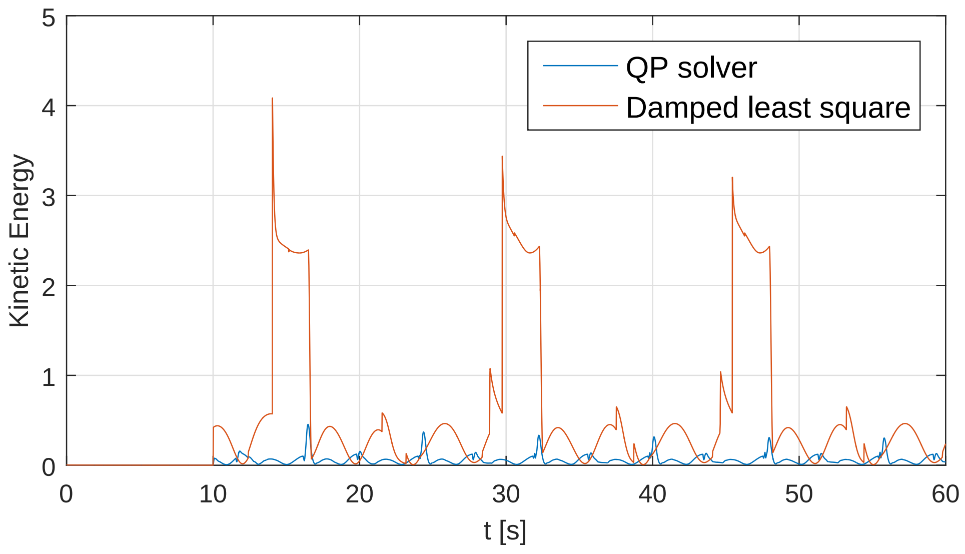


Figure 5: *Kinetic energy as a function of time for the QP solver and the damped least square based inverse kinematics solver of the Robot ToolKit.*

4.3 Results of the simulation

In order to assess the quality of the minimum kinetic energy algorithm, the results of the simulations will be compared to the ones found by the Robot ToolKit inverse kinematics solver. The Robot ToolKit solver is based on the damped least square method presented in section 1.1.2.

The kinetic energy as a function of time for the algorithm presented and the Robot ToolKit method is presented on figure 5. The kinetic energy is much lower with the minimum kinematic algorithm than with the standard algorithm of the Robot ToolKit method which does not try to minimize it.

In a Jacobian pseudo-inverse based method it is also possible to do null space optimization in order to minimize the kinetic energy [4]. It requires however the use of a hyper-parameter that can be difficult to set [1].

The trajectory of the end-effector of the robot during the simulation is represented in figure 6 along with the desired trajectory given by the parametric equation 29. The end-effector seems to follow well its desired motion during most of the trajectory but on the bottom left and bottom right of figure 6 there is a significant divergence. This error is caused by some of joints being too close to their limits, the joints velocities are reduced accordingly via the equation 10 to avoid the limits smoothly and we observe indeed no discontinuity in the trajectory.

Figure 7 shows that the error norm on the end-effector position varies a lot over time as there are configurations where joint limits are approached. The error norm with the Robot ToolKit's solver also fluctuates over time but with a much lower

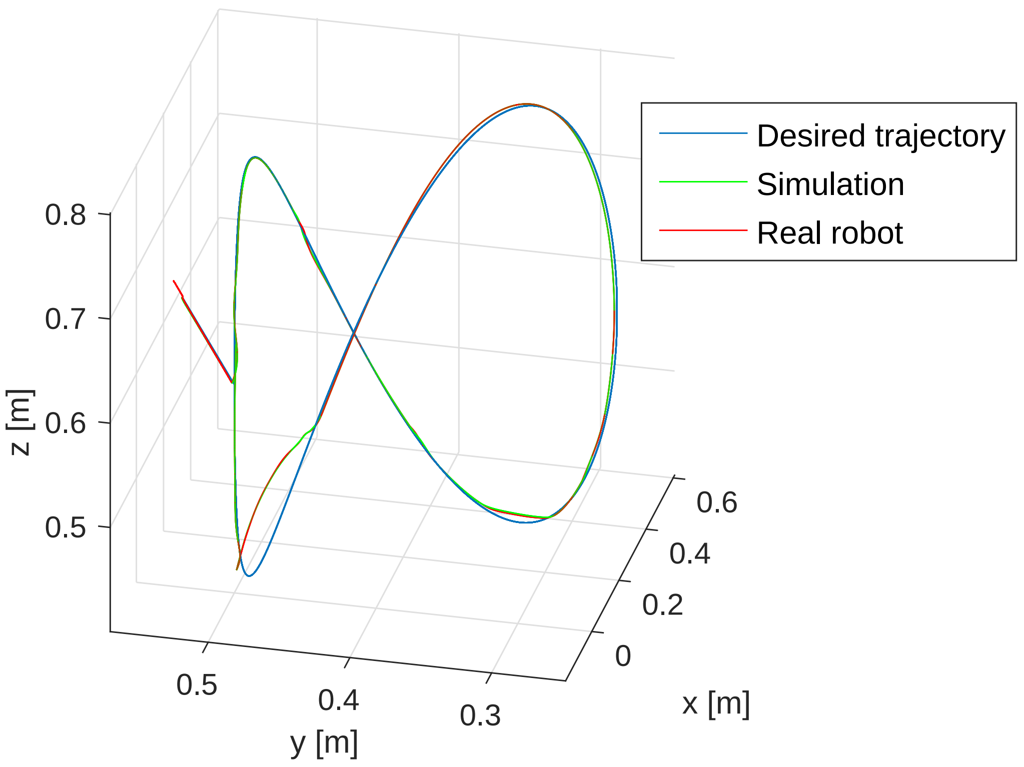


Figure 6: *First period of the trajectory of the end effector position.*

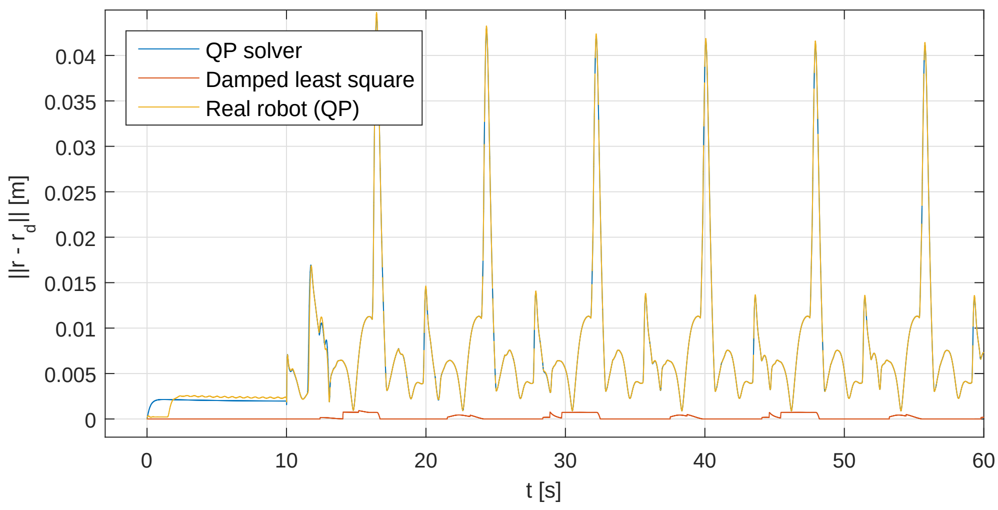


Figure 7: *Error norm on the position of the end-effector as a function of time for the QP and Robot ToolKit's damped least square based solvers in the simulator and with the QP solver on a real robot.*

amplitude.

At the peaks, the error norm with the QP solver reaches $4cm$ whereas the Robot ToolKit's imprecision is of the order of $1mm$.

One meaningful measure of the imprecision of the simulation can be the average error norm on r over time:

$$\Delta r := \frac{1}{T}\int_0^T \|r - r_d\| dt \tag{31}$$

The time of integration is set to $T = 60s$. It allows us to observe the impact of the end effector velocity on the precision as represented on the figure 8 that represents Δr as a function of k which tunes the speed of the simulation in equation 6. Δr increases with k as the motion is more difficult to follow the faster it is.

The method used to avoid the joints limits will have a greater influence on the motion if the velocity is large which is one of the explanation why Δr increases with k. If the velocity was infinitesimal, the joints limits have an impact only when they are actually reached.

The fact that the time-step is finite is also a source of positional error as the new θ at each time-step is found by forward Euler numerical integration:

$$\theta(t + \Delta t) = \theta(t) + \dot{\theta}(t) \cdot \Delta t \tag{32}$$

The error generated by this integration scheme is $\mathcal{O}(\Delta t)$ [3].

Figure 5 represents the kinetic energy over time for the QP based solver and the

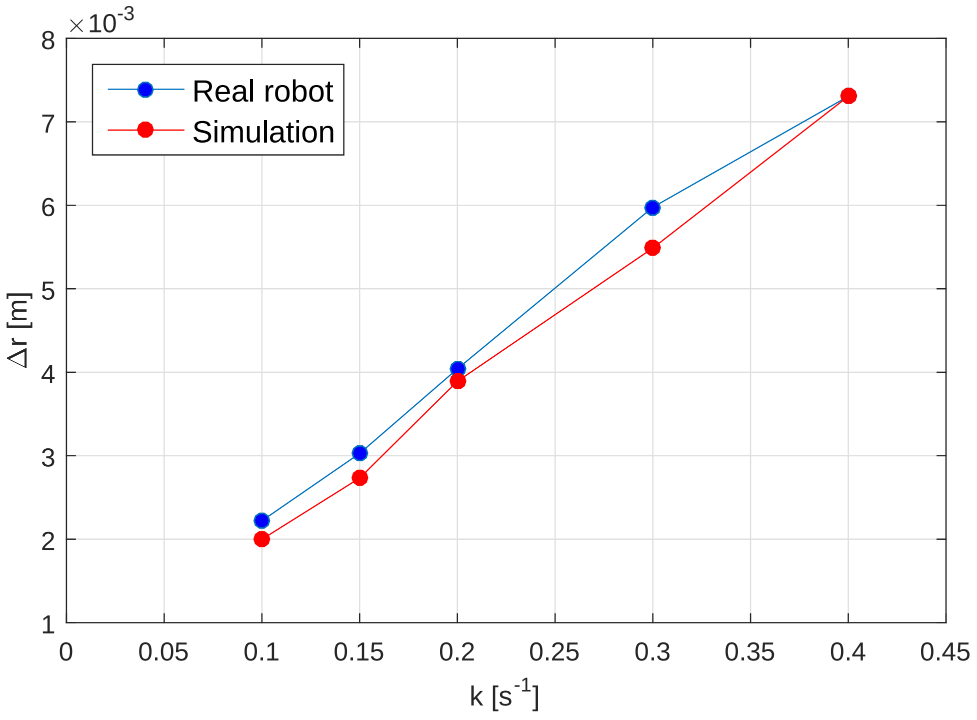


Figure 8: *Average norm of the error on the position for different values of k in simulation and with the robot. The average is done over time during the first minute of execution.*

solver used in the Robot ToolKit. The kinetic energy is lower with the QP solver as it tries to minimize it.

As represented on the figure 9, the avoidance of the joint limit is dealt in a smooth way with the QP solver whereas the joint stops abruptly when it reaches its limit with the Robot ToolKit's solver. The trajectories of the joints given by the QP solver and the Robot ToolKit being very different, the events presented on figure 9 do not happen at the same time. The Robot ToolKit and the QP solver minimize different quantity, respectively the joints velocity norm and the kinetic energy, which explains the large difference of the joints trajectories.

5 Implementation on a real robot

In order to test the presented algorithm in a real situation, the QP solver has been tested on a real KUKA LWR robot.

The robot follows the joints positions given by the solver as $\theta(t + \Delta t) = \theta(t) + \dot{\theta}(t) \cdot \Delta t$, $\dot{\theta}$ being given by the solver.

As shown in figure 6, the end-effector trajectory was almost the same as during the simulation, the robot being very precise.

Figure 8 shows that the error norm is on average not significantly larger with the real robot than during the simulation. It shows that algorithmic imprecisions are stronger than the technical imprecision of the KUKA. The average error norm is smaller than $1cm$ even with a $k = 0.4s^{-1}$.

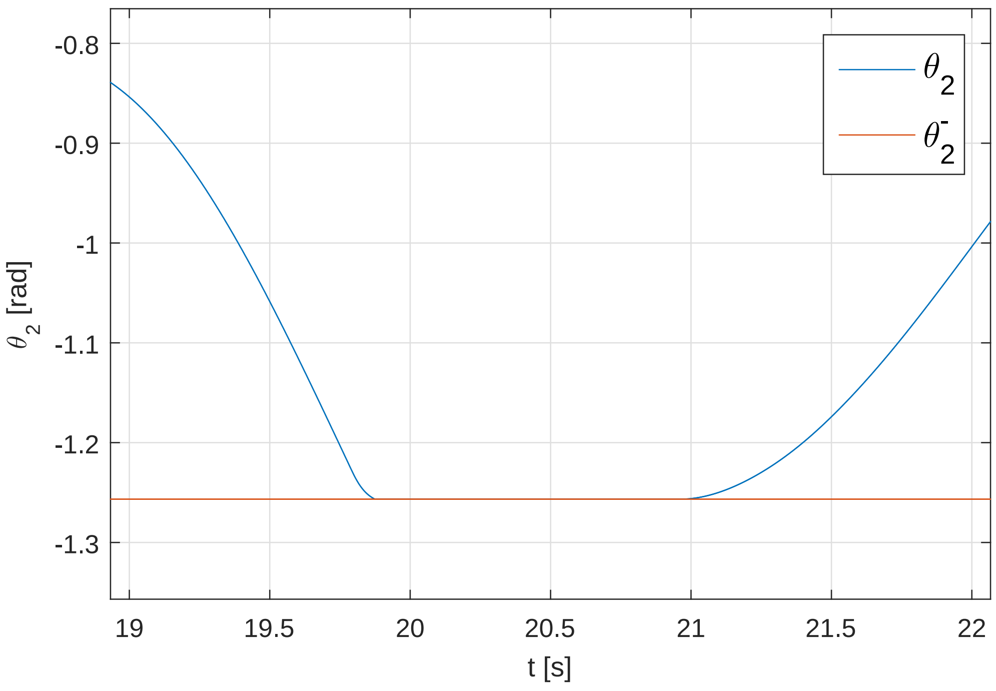
The fact that motion given by the QP solver is almost the same in reality as in the simulator seems to show that it is robust to noise.

6 Conclusion

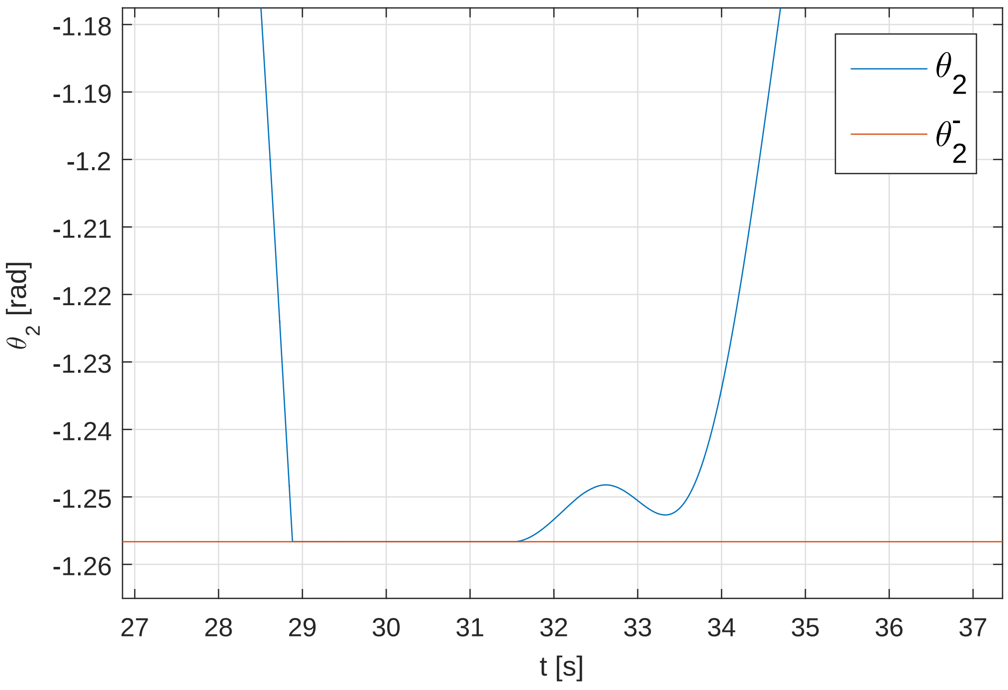
The goal of this semester project was to implement a quadratic programming based inverse kinematic method which is robust to singularities and minimizes the kinetic energy.

The algorithm has been coded in C++ and implemented in LASA's Robot ToolKit. The program has first been tested in simulator and then implemented in a real KUKA lightweight robot.

The quadratic based solver is not as accurate as the damped least squared based solver of the Robot ToolKit but it minimizes the kinetic energy in a natural way and it deals smoothly with the joints limits. The algorithm presented is well suited for real robots as the motion of the KUKA LWR was extremely close to what had been simulated. This algorithm is hence appropriate to use if the precision is not a



(a) QP solver



(b) Damped least square

Figure 9: Examples of the behaviour of the joints motion when a joint limit is reached with the QP solver and the Robot ToolKit's damped least square based solver.

major concern. A possible future development could be to improve the accuracy of the algorithm by using a better integrator for the numerical integration like Runge-Kutta's method.

References

[1] Y. Zhang, S. Ge and T. Lee, *A Unified Quadratic-Programming-Based Dynamical System Approach to Joint Torque Optimization of Physically Constrained Redundant Manipulators*, IEEE transactions on systems, man, and cybernetics (2004)

[2] S. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods* , University of California, San Diego (2009)

[3] J. Rappaz, M. Picasso, *Introduction à l'analyse numérique*, PPUR, Lausanne (2011)

[4] A. Liegeois, *Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms*, IEEE transactions on systems, man, and cybernetics (1977)

[5] W. Suleiman, *Inverse Kinematics: New Method for Minimum Jerk Trajectory Generation* (2014)

[6] J. Wang, Y. Li and X. Zhao, *Inverse Kinematics and Control of a 7-DOF Redundant Manipulator Based on the Closed-Loop Algorithm* Advanced Robotic Systems International (2010)

[7] O. Khatib. *A unified approach for motion and force control of robot manipulators: The operational space formulation* IEEE Journal of Robotics and Automation (1987)