Grasping Controller
Deliverable D7

Produced by WP4:
Giorgio Metta, Lorenzo Natale, Matteo Fumagalli, Ugo Pattacini

Grant Agreement number: 215805

Project acronym: CHRIS
Project title: Cooperative Human Robot Interaction Systems

Project start date: 01 March 2008

Funding Scheme: Small or medium-scale focused research project (STREP)

Date of latest version of Annex I against which the assessment will be made: April 2009

Project coordinator name, title and organization:
Prof Chris Melhuish, Director, Bristol Robotic Laboratory (BRL),
University of the West of England, Bristol

Tel: ++44(0)1173286334
Fax: ++44(0)1173282688
E-mail: Chris.Melhuish@brl.ac.uk

Project website address: http://www.chrisfp7.eu/

Contents

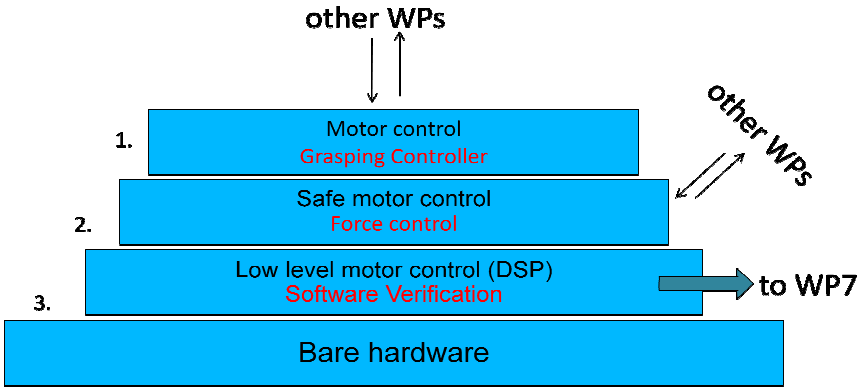

Figure 1. A schema of the software architecture as conceived in WP4 and interaction with other work-packages. Work in this deliverable implements parts of Layer 1 and 2.

1 Introduction

In this deliverable we describe the grasping behavior we implemented on the robot iCub. We focused on a scenario similar to the one considered and often described in the context of the CHRIS project: a human and a robot performing cooperative actions on objects placed on a table. As perception is not the scope of the project, we did not deal with aspects related to visual processing, like object detection, recognition and pose estimation. To solve these problems we exploited pre-existing software modules, whose descriptions are reported here only for completeness but should not be considered as a contribution of this deliverable.

Figure 1 shows a schematic representation of the layered software architecture as conceived within WP4. This deliverable focuses on motor control, and describes part of the system that implements Layer 1 and Layer 2 of this architecture. We have devised the grasping controller in two main parts. A *reaching* module, whose responsibility is to bring the hand of the robot close to the object and a *grasping* module that controls the movement of the fingers, and detects if and when the fingers apply enough force to the object surface. We studied force control with the aim to realize an impedance controller that guarantees physical safety during interaction. To guarantee interoperability with the other work-packages particular care was taken to define and implement a clear interface to the capability developed within WP4 and described in this deliverable.

The remainder of this document is organized as follows: Section 2 quickly describes the visual process on the robot. Section 3 provides details about the *reaching* module, including the techniques performing the inverse kinematics and computing a smooth trajectory to the target. Section 4 describes the *grasping* module and the algorithm that allows discriminating between successful and unsuccessful grasps. Finally, in Section 5 we describe the force and impedance control of the robot.

2 Visual Processing

The earliest stage of the visual processing system consists in detecting regions in the (visual) space towards which directing gaze. A commonly adopted solution is to employ a set of filters (each tuned to features like colors, orientations and motion) and compute the difference of the output of filters at different scales to obtain a saliency map (often filters are chosen to approximate the response of neurons in the primary cortex of primates). This saliency map is then searched for local maxima which correspond to “interesting” regions in the visual scene. The gaze of the robot is finally directed

towards these saliency regions using a certain criteria (in our case using a "winner-take-all" approach, but other strategies like random walk could be easily thought). Each feature map can be given more importance, in our case we decided to give more priority to motion so that moving objects are more likely to attract the attention of the robot. Alternatively the Spikenet object recognition software can provide a top-down cue which drives the attention of the robot towards known objects. The gaze control module controls the motor of the head to bring salient regions at the center of the cameras.

A lower level segmentation algorithm groups together areas in the images that have uniform color and extract the center of the area that is closer to the center of the image. The process described above exploits only visual information from one camera, and extract only the location of the target in image coordinates (u,v). Since the extraction of 3D information is notoriously difficult and imprecise we decided to take a pragmatic approach and compute the missing information from the assumption that all relevant objects lay on a table whose height is known a priori. With this assumption the visual processing module can compute the Cartesian position (x,y,z) of the target with respect to a known reference frame.

3 Reaching: the Cartesian Controller Interface

Given the Cartesian position of target object reaching is performed in two separate stages/modules (Figure 2). The first stage employs **a non linear optimization technique** to determine the arm joints configuration q_d that achieves the desired pose (i.e. end-effector position and orientation). The second stage consists in a **biologically inspired controller** that computes the velocity $\dot{q}$ of the motors to produce a human-like quasi-straight trajectory of the end-effector.

3.1 Solver module

We consider the general problem of computing the value of joint encoders $q^* \in R^n$ that achieves a given position $x_d \in R^3$ and orientation $\alpha_d \in R^3$ of the end effector, and at the same time, satisfies a set of given constraints expressed as inequalities.

Formally this problem can be expressed as:

$$q^* = \arg\min_{q \in R^n} \left(\left\| \alpha_d - K_\alpha(q) \right\|^2 + \lambda \cdot (q_{rest} - q)^T W (q_{rest} - q) \right)$$
$$\text{s.t.} \begin{cases} \left\| x_d - K_x(q) \right\|^2 < \varepsilon \\ q_L < q < q_U \end{cases}, \quad (1)$$

where K_x and K_α are the forward kinematic functions that respectively compute position and orientation of the end-effector from the joint angles q; q_{rest} is a preferred joint configuration, W is

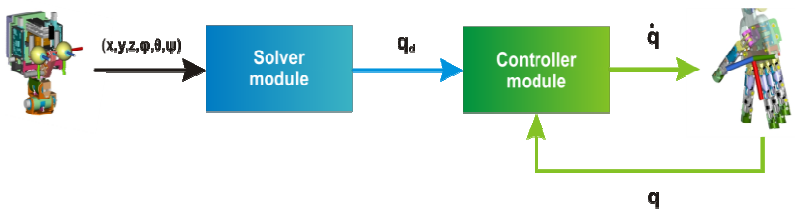

Figure 2. Diagram of the Cartesian controller.

a diagonal matrix of weighting factors, λ is a positive scalar (<1) and ε a *small* number (in the range of $10^{-4} \div 10^{-5}$). Moreover, the solution to problem (1) has to comply with a set of additional constraints: for example, we require that the solution lies between lower and upper bounds ($q_U, q_L \in R^n$) of physically admissible values.

In our case the joints vector has 10 components (7 joints for the arm, 3 joints for the torso) and we choose the value of q_{rest} so that the torso of the robot is as close as possible to the vertical position. We propose to use an interior point optimization technique to solve the problem (1), in particular we use *IpOpt* [1], a public domain software package designed for large-scale nonlinear optimization.

This approach has the following advantages:

1. *Quick convergence*. *IpOpt* is fast enough to be employed in real-time as demonstrated below.
2. *Automatic handling of singularities and joint limits*. This technique automatically deals with singularities in the arm Jacobian and joint limits, and was able to find solutions in virtually any working conditions;
3. *Tasks hierarchy*. The task is split in two subtasks: the control of the orientation and the control of the position of the end-effector. These subtasks are given different priorities, for example in our case the control of the position has higher priority with respect to the orientation subtask (former is treated as a nonlinear constraint and thus evaluated before the cost);
4. *Description of complex constraints*. It is easy to add new constraints as linear and/or nonlinear inequalities either in task or joint space. In the case of the iCub, for example, we add a set of constraints that avoid breaking the tendons of the shoulder (for practical reasons the tendons of the shoulder do not permit to cover the complete convex hull of the shoulder's movements within the joint bounds).

3.2 Controller module

We now need to determine smooth velocity profiles in the joint space which steers the arm from the current posture q to the final configuration q^*, while at the same time ensuring that the joints lay within well defined limits. This can be obtained by applying the Multi-Referential Dynamical Systems approach [3], in which two dynamical controllers, one in joint space and one in task space, evolve concurrently; the coherence constraint between the two tasks is enforced with the Lagrangian multipliers method and can be used to modulate the relative influence of each controller (i.e. to avoid joint angles limits). The advantage of such a redundant representation of the movement is that a quasi-straight trajectory profile can be generated for the end-effector in the task space reproducing a human-like behavior [4,5], while retaining converge property and robustness against singularities (the method can be cast back to the DLS algorithm).

In [3] the two controllers are implemented with VITE models [2], which approximate the neural signals commanding a pair of agonist-antagonist muscles (the biological analogous of a mass-spring-damper system). The model behavior is regulated by the second order differential equation:

$$\ddot{x} = \alpha \cdot \left(\beta \cdot \left(x_d - x \right) - \dot{x} \right), \tag{2}$$

where α is the damping factor and $\alpha \cdot \beta$ is the stiffness.

Aside from the connection to biological evidences, a second important merit of this approach is that the VITE model is described by a compact and time-invariant differential equation which makes the controller implementation straightforward. On the other hand, the specific choice of a second order dynamic system entails a couple of major disadvantages when applied to the control of a robotic limb:

1. There exists a non-trivial map between the desired point-to-point movement time and the values of the controller's parameters α, β. This aspect complicates the design of the controller whose time response needs to be modified on the fly;
2. More notably, the resulting velocity profiles become less human-like as the required execution time becomes shorter. Actually, when a fast response is requested, trajectories approach an exponential response (typical of a first order dynamical system), irrespectively of how damping and stiffness are tuned; therefore, the corresponding velocities are no longer bell-shaped, having a steep acceleration at the beginning followed by a slow decay. The reason is that a second order system just cannot reproduce the smoothness typical of biological motion [5] (for example it does not impose zero acceleration at starting point). As a result fast movements tend to be too jerky and produce unwanted vibrations.

To overcome these problems, we maintain the multi-referential nature of the controller and replace the VITE systems with more complex controllers which reproduce a trajectory that resemble a minimum-jerk profile both in joint and task space.

Movements are still represented and controlled in multiple frames of reference but maintain a smooth (bell shaped) velocity profile. We took inspiration from the feedback formulation of a minimum-jerk trajectory described in [6], where a third order linear time-variant differential equation is derived:

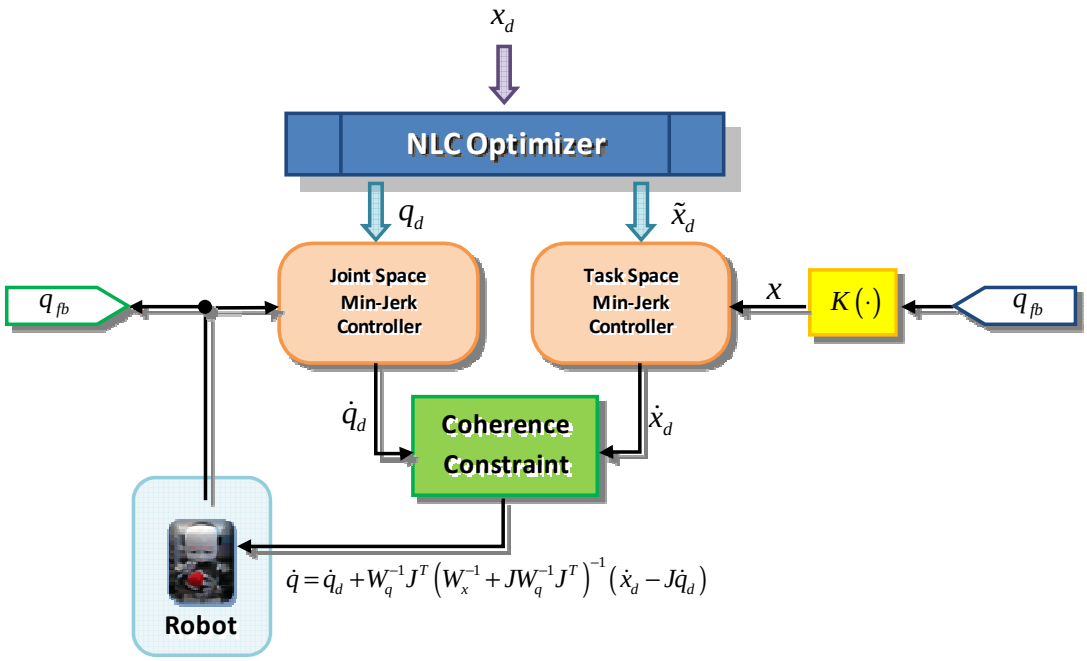

Figure 3. Multi-Referential controllers diagram.

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dddot{x}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -\frac{60}{(T-t)^3} & -\frac{36}{(T-t)^2} & -\frac{9}{T-t}\end{bmatrix}\cdot\begin{bmatrix}x\\ \dot{x}\\ \ddot{x}\end{bmatrix}+\begin{bmatrix}0\\ 0\\ \frac{60}{(T-t)^3}\end{bmatrix}\cdot x_d \tag{3}$$

The system in (3) produces the well known minimum-jerk shape depicted in Figure 4 (red lines) and described by the usual fifth-order polynomial function:

$$x(t)=x_0+(x_d-x_0)\cdot\left(10\cdot\left(\frac{t}{T}\right)^3-15\cdot\left(\frac{t}{T}\right)^4+6\cdot\left(\frac{t}{T}\right)^5\right) \tag{4}$$

The problem of the feedback formulation in (3) is that it contains time-variant coefficients whose value become infinite when *t* approaches the execution time *T*. To overcome this issue, we decided to employ a time-invariant third order system whose parameters are tuned to better approximate a minimum jerk trajectory (in other words we sought the linear third order differential systems that is the best time-invariant version of (3) that minimizes the same jerk measure over the interval [*0, T*]). Formally, we started from the parametric equation of the trajectory expressed in the form:

$$x(t)=C_1\cdot e^{\lambda_1 t}+C_2\cdot e^{\lambda_2 t}+C_3\cdot e^{\lambda_3 t}+x_d \tag{5}$$

that is a particular solution of a stable third order differential system with three independent real negative poles λ_i (we want to avoid damped resonant terms, since we require a monotonic trend to the target). The coefficients C_i can be determined for the special case $x_d=1$ and $x(0)=0$ (without any loss of generality) by imposing the following initial conditions:

$$\begin{cases}x(0)=0;\\ \dot{x}(0)=0;\\ \ddot{x}(0)=0;\end{cases}\Rightarrow\mathbf{C}=\begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\ \lambda_1 & \lambda_2 & \lambda_3\\ \lambda_1^2 & \lambda_2^2 & \lambda_3^2\end{bmatrix}^{-1}\cdot\begin{bmatrix}-1\\ 0\\ 0\end{bmatrix}\Rightarrow\mathbf{C}=\begin{bmatrix}-\frac{\lambda_2\lambda_3}{(\lambda_1-\lambda_2)\cdot(\lambda_1-\lambda_3)}\\ \frac{\lambda_1\lambda_3}{(\lambda_1-\lambda_2)\cdot(\lambda_1-\lambda_3)}\\ -\frac{\lambda_1\lambda_2}{(\lambda_1-\lambda_2)\cdot(\lambda_1-\lambda_3)}\end{bmatrix}. \tag{6}$$

Therefore, defined $M(t)$ the measure of the jerk accumulated up to time *t*:

$$M(t)=\int_0^t\dddot{x}^2(\tau)d\tau, \tag{7}$$

we seek for a solution to the following minimization problem:

$$\boldsymbol{\lambda}=\begin{bmatrix}\lambda_1\\ \lambda_2\\ \lambda_3\end{bmatrix}=\arg\min_{R^3}M(\infty)\quad s.t.\quad\begin{cases}\lambda_i<0,\ \forall i=1,2,3\\ \lambda_i\neq\lambda_j,\ \forall i\neq j\\ x(1)\geq 1-\varepsilon_1\end{cases}. \tag{8}$$

The first constraint in (8) imposes that the system is stable, while the second constraint requires the independence of all the roots, which is strictly necessary to solve the linear system for the coefficients C_i. This latter nonlinear bound can be implemented by resorting to a suitable set of continuously differentiable boundary functions such as:

$$\begin{cases} e^{-\frac{(\lambda_1-\lambda_2)^2}{\sigma^2}} \le \varepsilon_2 \\ e^{-\frac{(\lambda_1-\lambda_3)^2}{\sigma^2}} \le \varepsilon_2 \\ e^{-\frac{(\lambda_2-\lambda_3)^2}{\sigma^2}} \le \varepsilon_2 \end{cases} \tag{9}$$

that, depending on the values assigned to σ and ε_2, guarantees that the roots are non coincident (e.g. $\sigma=10^{-3}$ and $\varepsilon_2=0.1$ imply $\left|\lambda_i-\lambda_j\right|>10^{-3}$).

Finally the third constraint in (8) forces the solution to reach the steady-state value of 1 with a "rate" specified by the parameter ε_1. Without this lower bound on $x(1)$ any possible monotonically increasing function $x(t)$ would be allowed, even functions with very slow time constants. In other words by setting the parameter ε_1 we are able to tune the final execution time which, in our case, will be as close as possible to 1.

We set $\varepsilon_1=0.1$ ($x(1)\ge 0.9$) and ran an optimization algorithm to solve (8) (we used the interior-point optimization algorithm as well). Figure 4 compares the trajectory (position, velocity and total jerk) of the ideal minimum-jerk model against the one obtained with the time-invariant system derived with our approach. As expected the third-order system just approximates the ideal minimum-jerk trajectory, having in particular a slightly faster onset followed by slower convergence to the steady state. At the same time, however, it provides a very good compromise between smoothness and simplicity of implementation.

Once the roots λ_i are known, it is straightforward to compute the elements of the dynamic matrix A and input matrix B of the system in canonical form:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = A \cdot \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + B \cdot x_d$$
$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ a & b & c \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}. \tag{10}$$

Indeed, the elements a, b and c can be resolved by comparing the coefficients of the characteristic polynomial of the differential equation and the coefficients of the characteristic polynomial of A:

$$(\lambda-\lambda_1)\cdot(\lambda-\lambda_2)\cdot(\lambda-\lambda_3) \equiv \det(\lambda I - A) = \lambda^3 - c\lambda^2 - b\lambda - a. \tag{11}$$

The element d is completely determined from convergence reasons and equal to $-a$, noticing that for $t=\infty$ it holds:

$$\begin{cases} \dddot{x}(\infty) = a\cdot x(\infty) + b\cdot \dot{x}(\infty) + c\cdot \ddot{x}(\infty) + d\cdot x_d \\ \dddot{x}(\infty) = \ddot{x}(\infty) = \dot{x}(\infty) = 0 \\ x(\infty) = x_d \end{cases} \tag{12}$$

Following these considerations from the roots λ_i we derived the equivalent dynamical system rescaled for a generic execution time T:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{150.832}{T^3} & -\frac{85}{T^2} & -\frac{15.969}{T} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \frac{150.832}{T^3} \end{bmatrix} \tag{13}$$

The time rescaling can be performed knowing that the generic derivative of order *n* of the rescaled function $x(t/T)$ is $x^{(n)}(t/T)/T^n$.

The system response in $t=T$ is equal to the 90% of the steady-state value as expected, and the transient can be considered extinguished for $t \geq 1.5T$. Clearly only one parameter modulates the speed of the system, with a linear relationship between the gain *T* and the real execution time as required.

Figure 5 shows the Bode diagram for the frequency response of the identified system with different values of the gain *T*: the system has been previously discretized with a sampling time of 10 ms, which is the normal period used to run the controller.

We also verified if a 4th order could provide a trajectory closer to minimum-jerk. Clearly, this is the case as a new degree of freedom is introduced by the fourth root and a better trade-off becomes achievable. Unfortunately we found that this solution has a much lower stability margin. Since one of the requirements is to have good system response to fast trajectories, we eventually decided to discard higher order models.

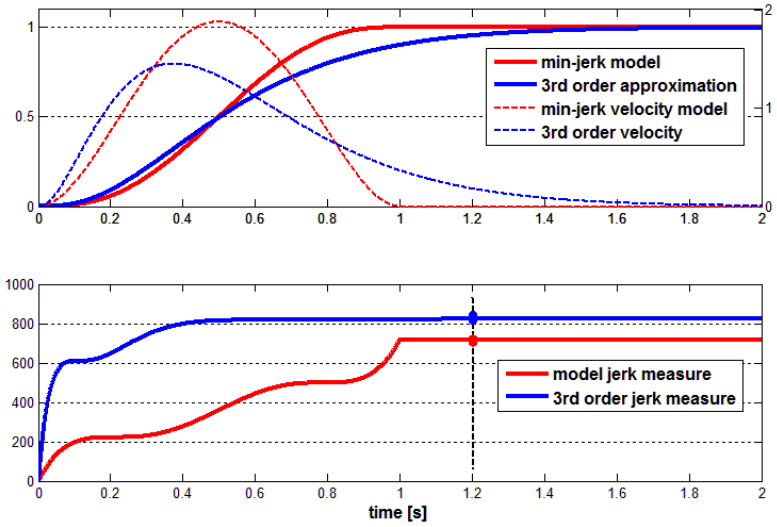

Figure 4. Comparison between the 3rd order dynamical system found through the minimization and the minimum-jerk model.

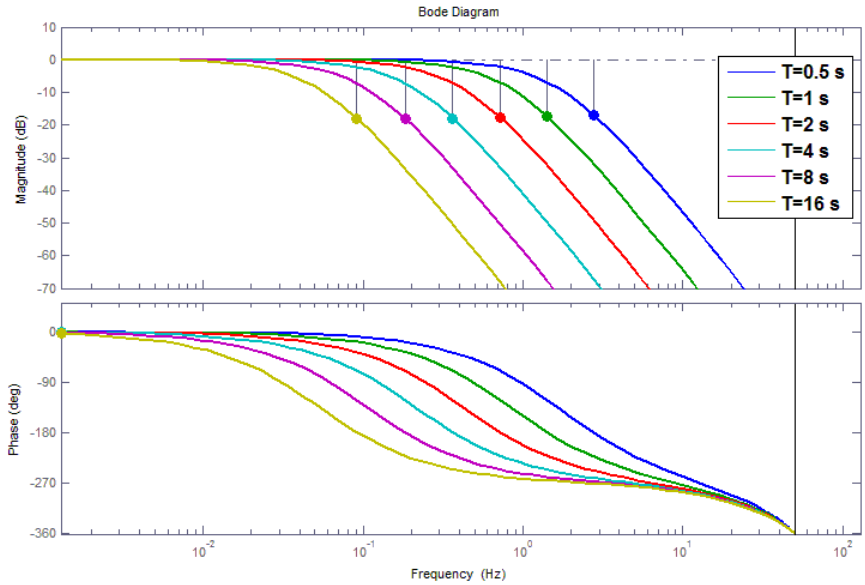

Figure 5. Bode diagram of the 3rd order system for several values of T: gain margins are also displayed.

3.3 Controller module: the algorithm

Similarly to [3] the algorithm has three steps.

1. At any step each controller receives the desired target found by the optimization process, the current value of q_{fb} (and $x_{fb} = K(q_{fb})$), as in Figure 3 and computes the corresponding velocity profile by integrating at the time instant *t* equation (10), which holds both in the joint and operational space:

$$\begin{bmatrix} x \\ \dot{x}_t^d \\ \ddot{x} \end{bmatrix} = \int_0^t \left(A \cdot \begin{bmatrix} K(q_{fb}) \\ \dot{x} \\ \ddot{x} \end{bmatrix} + B \cdot \tilde{x}_d \right) dt, \tag{14}$$

$$\begin{bmatrix} q \\ \dot{q}_t^d \\ \ddot{q} \end{bmatrix} = \int_0^t \left(A \cdot \begin{bmatrix} q_{fb} \\ \dot{q} \\ \ddot{q} \end{bmatrix} + B \cdot q_d \right) dt. \tag{15}$$

2. Since the two controllers evolve independently, the two trajectories are unlikely to satisfy the kinematic constraint given by $\dot{x}^d = J \cdot \dot{q}^d$, being J the geometric Jacobian of the forward kinematic map $K(\cdot)$. The *coherence* between the two trajectories is thus enforced by computing the joint velocities that solve the following minimization problem:

$$\min_{\dot{q}_t, \dot{x}_t} \frac{1}{2} \left(\left(\dot{q}_t - \dot{q}_t^d \right)^T W_q \left(\dot{q}_t - \dot{q}_t^d \right) + \left(\dot{x}_t - \dot{x}_t^d \right)^T W_x \left(\dot{x}_t - \dot{x}_t^d \right) \right) \quad s.t. \quad \dot{x}_t = J \dot{q}_t \tag{16}$$

By applying the Lagrangian multipliers method we can compute a closed form solution:

$$\dot{q}_{t+1} = \dot{q}_t^d + W_q^{-1} J^T \left(W_x^{-1} + J W_q^{-1} J^T \right)^{-1} \left(\dot{x}_t^d - J \dot{q}_t^d \right) \tag{17}$$

With W_q and W_x appropriate semi-definite diagonal matrices as defined in step 3.

3. The matrices W_q and W_x can be used to give different importance to the joint or task space constraints. This can be exploited for example to implement a mechanism for **joints limits**

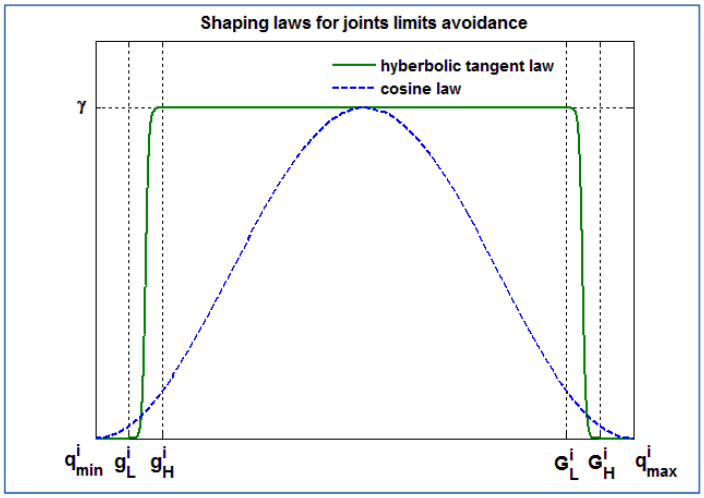

Figure 6. Comparison between the implemented shaping law for the joints limits avoidance with the guard ratio ρ=0.25 and the law in [3] based on a cosine map.

avoidance. Normally W_q and W_x are set so that the task space constraint has higher priority and the arm follows a straight path. When one of the joints approaches a limit, however, W_q is increased so that the joint space constraint assumes more importance. Formally when the ith joint angle is within the bounds the corresponding weight $w_q^i \in W_q$ is close to zero; conversely, when the arm gets closer to one of the joint limits, the element w_q^i become larger, until, eventually, the ratio w_x / w_q^i goes to zero. In the latter situation, since the controller evolves in a convex space, the arm is guaranteed to respect the joint limits: for this reason it is crucial that the controllers produce monotonically increasing trajectories without overshoots.

In [3], this modulation is achieved by imposing that at each time instant the following shaping relation holds:

$$\frac{w_x}{w_q^i} = \frac{1}{2}\gamma\left(1-\cos\left(2\pi\cdot\frac{q^i - q^i_{\min}}{q^i_{\max} - q^i_{\min}}\right)\right), \tag{18}$$

Where γ is a normalization constant typically around 0.01 [3].

To better exploit the whole arm workspace it is advisable to assign high priority to the Cartesian controller in a portion of the joint space that is as large as possible. For this purpose we adopted a different weighting policy, made of a flat region connected with hyperbolic tangent functions whose decay rate is much steeper than the original cosine law, as illustrated in Figure 6:

$$\frac{w_x}{w_q^i} = \begin{cases} \frac{1}{2}\gamma\cdot\left(1+\tanh\left(10\cdot\frac{q^i-\bar{g}^i}{d^i}\right)\right), & g_L^i < q^i < g_H^i \\ \frac{1}{2}\gamma\cdot\left(1-\tanh\left(10\cdot\frac{q^i-\bar{G}^i}{d^i}\right)\right), & G_L^i < q^i < G_H^i \\ \gamma, & g_H^i < q^i < G_L^i \\ 0, & \text{elsewhere} \end{cases}, \tag{19}$$

with:

$$\begin{cases} d^i = \dfrac{\rho\cdot\left(q^i_{\max} - q^i_{\min}\right)}{4} \\ g_L^i = q^i_{\min} + d^i;\ g_H^i = g_L^i + d^i;\ \bar{g}^i = \dfrac{g_L^i + g_H^i}{2} \\ G_H^i = q^i_{\max} - d^i;\ G_L^i = G_H^i - d^i;\ \bar{G}^i = \dfrac{G_L^i + G_H^i}{2} \end{cases}. \tag{20}$$

A possible strategy for modulating the weights is setting W_x^{-1} equal to identity matrix and W_q^{-1} equal to the right-hand side of (19).

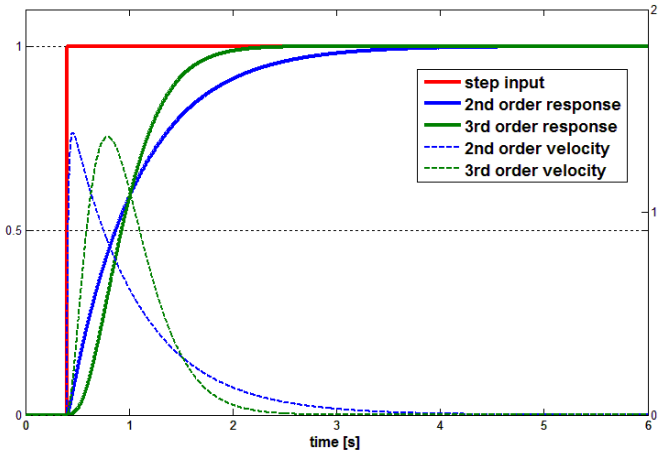

Figure 7. Comparison between step responses of VITE's (blue) and minimum-jerk (green) controllers. The 3rd order system provides the same velocity peak, but has faster convergence and an (almost) bell-shaped velocity profile.

3.4 Results

The first important result achieved by our controller is in terms of performances: from Figure 7 it is clear that minimum-jerk controllers can provide, especially for fast trajectories, smooth velocity profiles that are more similar to the desired human-like prototypes if compared to the profiles imposed by the VITE models.

Figure 8 shows the Cartesian position of the end-effector while tracking a desired pose in the operational space; in this particular example 10 degrees of freedom are controlled (7 for the arm along with the pitch, the roll and the yaw joints of the torso): one cycle of the lemniscate-shaped desired trajectory in front of the robot frontal plane was executed in 20 seconds, whereas the time control gain *T* for point-to-point movements was set to 0.5 seconds.

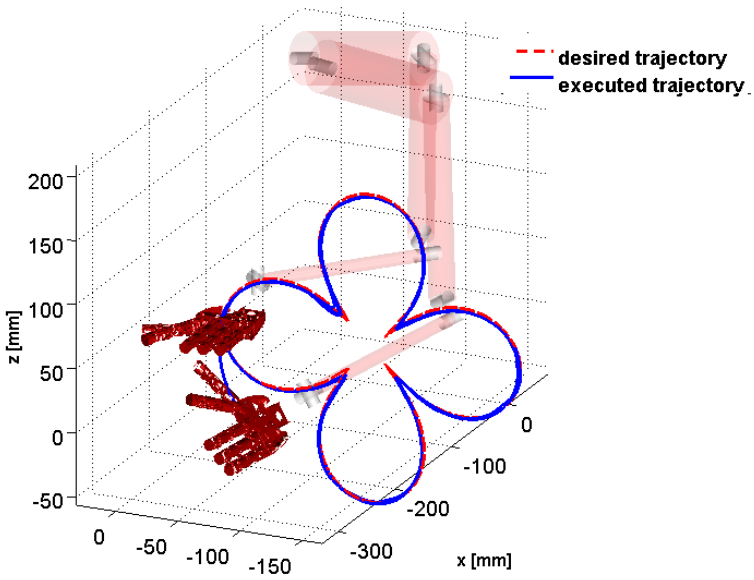

Figure 8. Tracking a desired trajectory with a lemniscate shape in the operational space. For better understanding the figure shows two configurations of the arm: the lower one depicts the starting pose, whilst the upper one shows the commanded hand orientation during the task.

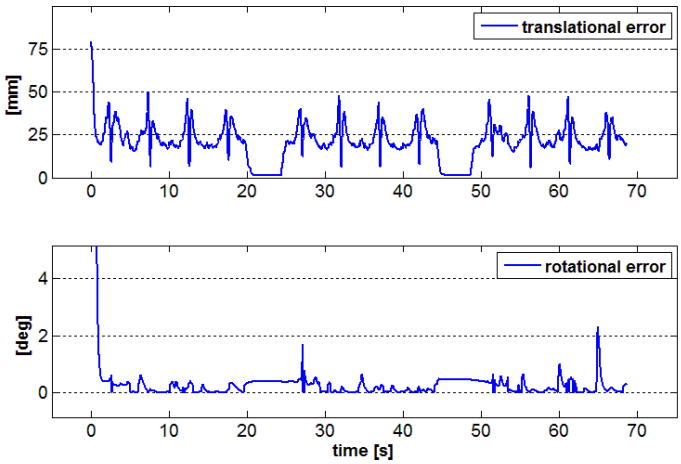

Figure 9. Tracking errors.

To verify speed and accuracy of the tracking Figure 9 reports a plot of the errors during three iterations around the whole lemniscate shape: after an initial movement, required to catch up with the target, the position of the end-effector remains constrained within a ball of radius 5 cm centered at the trajectory point, while the error is virtually zero at the end of the cycles; similar considerations apply for the rotational error here computed as the Euclidean norm of the difference between the desired rotational axis (whose magnitude represents the angular amount of rotation) and the actual one.

Furthermore, as discussed previously and demonstrated in Figure 10, *IpOpt* turns out to be fast enough to be used to track a target in real time: during tracking the average computation time is below 20 ms (on a multi-core Intel (R) Xeon with 2.27 GHz of clock frequency). This allows performing tracking with a constant solver rate of 33 Hz and a controller rate of 100 Hz. Interestingly, only a small latency is experienced at the beginning of the first cycle when the starting position of the end-effector is far from the target: this is expected since the optimization algorithm starts from an initial "guess" that is far from the solution and takes a longer convergence time. However, the initial overhead is acceptable (<50 ms), and does not cause a significant degradation of the real-time performance.

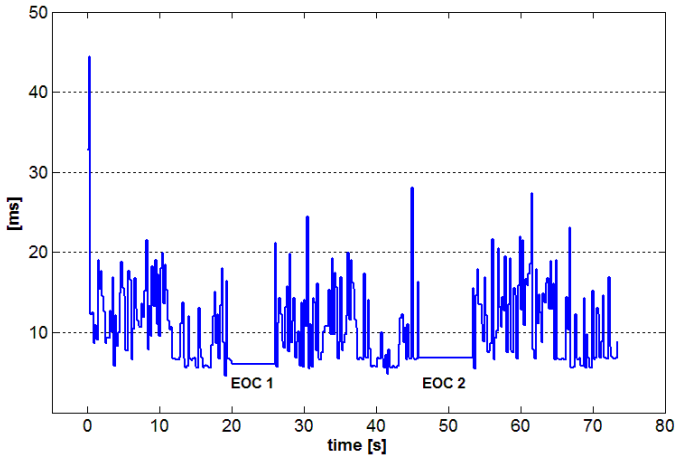

Figure 10. A plot of solver computation time during tracking of the lemniscates-shaped trajectory: the end of first and second cycle (respectively EOC1/EOC2) is visible in the figure.

3.5 Cartesian Interface

To simplify use (and re-use) of our controller we extended *YARP*[1] to support the Cartesian controller we realized. In brief this consisted in extending the existing joint level motor control interfaces to include task space control of any robotic structure. The purpose was twofold: (1) achieve better modularity and (2) hide the implementation details of the controller behind a set of simple interfaces (e.g. *go_to_pose*(), *get_pose*(), *set_trajectory_time*(), …)[2].

By separating the trajectory generator from the solver we were able to run the controller directly on the robot (i.e. on the PC104 hub) and reduce latencies associated with the network. We thus obtained a considerable gain in controller performance. Running the controller on board of the PC104 allowed to obtain a factor of about 50% performance increase with respect to the standard *YARP* module accessing the iCub through the network. We adopted a client/server architecture (Figure 11). The client exports the Cartesian Interface as it is seen by the user. Accessing the client at this level is as easy as creating a C++ object and calling its methods. Through YARP the client implements the request of the user by dispatching them to the Server and, hidden to the user, the Solver object. The Server runs locally on the robot and implements the core of the control based on the Multi-Referential approach (since the controller sends velocity commands to the low-level hardware, latencies at this point are crucial). The hidden part is represented by the *S*olver which computes the set points for the Server using *IpOpt*; since this process is computationally expensive, it is advisable to run the Solver on a separated, powerful machine (as the output of the Solvers are set-points, latencies at this level are less critical).

Furthermore, another layer of modularity was achieved through the realization of the *iKin* library. This is a general purpose kinematics library that can be configured to replicate the forward kinematics of any serial link chain; this makes straightforward to re-use the Cartesian Controller on different robotic platforms such as the ones available within the CHRIS project, i.e. *BERT1* and *HRP2*.

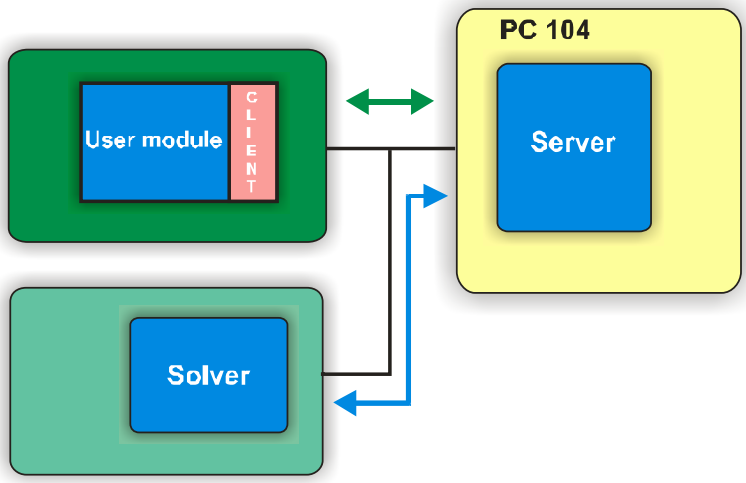

Figure 11. The modular Cartesian Interface architecture.

[1] *YARP*, Yet Another Robotic Platform, i.e. the software middleware used on the *iCub*.
[2] A complete list of available methods is available on the web: http://eris.liralab.it/yarpdoc/dd/de6/classyarp_1_1dev_1_1ICartesianControl.html.

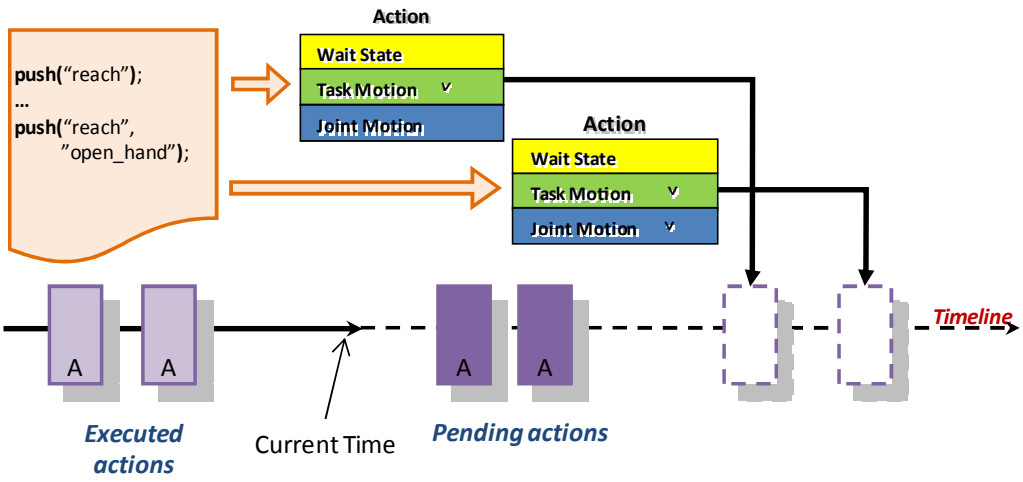

Figure 12. The Action Primitives Library.

3.6 Discussion: inverting the kinematics without the kinematics

The technique we have discussed so far requires knowledge of the direct kinematics and Jacobian of the manipulator. The Jacobian of the manipulator in particular is used by *IpOpt* to solve the minimization problem:

$$q^* = \arg\min_{q\in R^n} \left(\left\| \alpha_d - K_\alpha(q) \right\|^2 + \lambda \cdot (q_{rest} - q)^T W (q_{rest} - q) \right)$$
$$\text{s.t.} \begin{cases} \left\| x_d - K_x(q) \right\|^2 < \varepsilon \\ q_L < q < q_U \end{cases}, \tag{21}$$

while the minimum-jerk controller (Figure 3) requires direct kinematics and Jacobian to compute the task space trajectory and enforce the coherence constraints.

On the robot iCub we have applied these techniques using the direct kinematics and Jacobian derived from the CAD model of the robot. However the software is written is such a way that the functions computing $K_\alpha(q) / K_x(q)$ and $J(\cdot)$ can be specified by the user and can be replaced with no cost with a learned model (e.g. a neural network).

4 Grasping: the Action Primitives Library

We have developed a library that relies on the *YARP* Cartesian Interface and realizes a further abstraction layer that expose to the user a collection of action primitives (such as *reach*(), *grasp*(), *tap*(), …) along with an easy way to combine them together to form higher level actions and eventually execute more sophisticated tasks.

Central to the Action Primitives library is the concept of *action*. An action is a "request" for execution of three tasks (as depicted in Figure 12):

1. It can ask to steer the arm to a specified pose, hence performing a motion in the task space;
2. It can command the execution of some predefined finger sequences in the joint space and identify it with a tag;
3. It can ask the system to wait for a specified time interval;

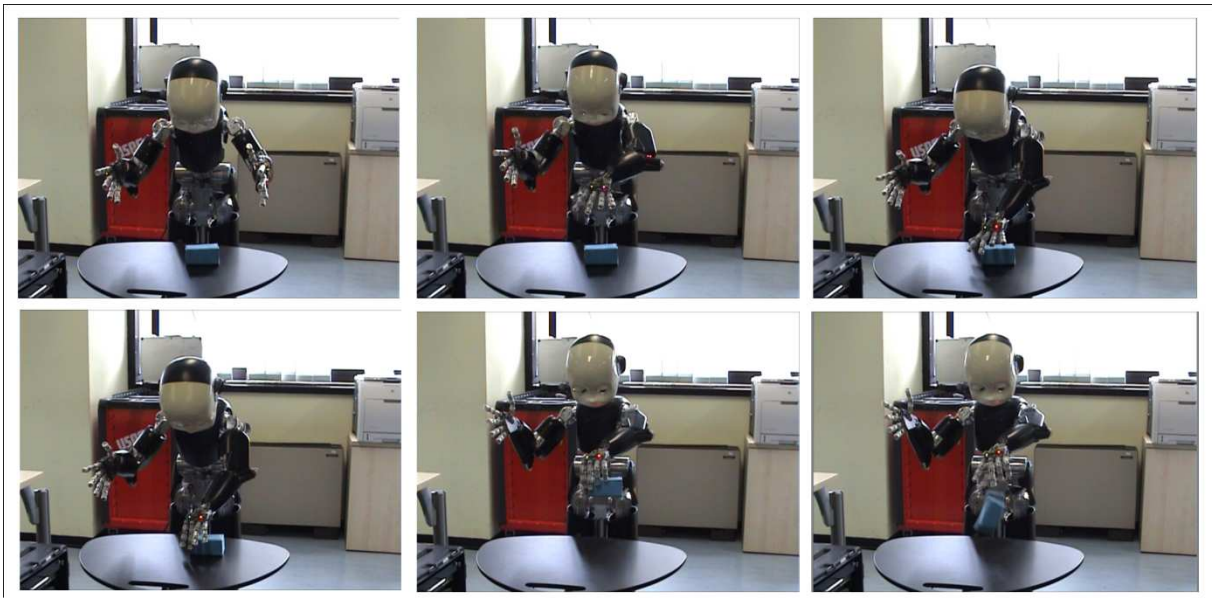

Figure 13. A grasping sequence. From top-left to bottom-right, an object is placed on the table, the robot moves the hand above the object and closes the hand. When a successful grasp is detected the robot lifts the object and eventually drops it.

Besides, the library offers the possibility to specify if the action involves the execution of a task of type 1 simultaneously with a task of type 2. Action requests are stored in an *action queue* and served with a First In First out policy. The idea is to provide a mechanism to program and execute sequences of actions as combinations of movements of the arm (type 1), movements of the fingers (type 2) or coordinated movement of both arm and fingers (type1+type2).

An important aspect to point out is that at the moment the finger positions that realize the grasp actions should be known (and tuned) in advance. However we are planning to include a learning stage that eventually will allow to automatically adapt the motion of the fingers to novel objects. In Figure 13 we report a sequence that shows the robot grasping an object.

4.1 The detection problem

In absence of tactile feedback we implemented an algorithm that performs contact detection using the springs mounted in the distal phalanges of the hand. Due to the elastic coupling between the phalanges of the fingers[3], the fingers passively adapt when they encounter an obstacle (i.e. when they touch the surface of an object). The amount of "adaptation" can be indirectly estimated from the encoders on the joints of the fingers. In other words, the idea is to measure the discrepancy between the finger motion in presence of external obstacles (e.g. objects or the other fingers) and the one that would result in normal operation (in absence of obstacles/free movement). In a calibration phase we estimate the (linear) relationship between the joints of the fingers in absence of contact. In normal operation, we detect contact by comparing how much this model fit the current encoder readings.

[3] On the iCub fingers are made up of three phalanges that we call proximal, middle and distal. With the exception of the ring and little fingers, the middle and distal phalanges are mechanically coupled with an elastic element and actuated by a single motor. In the ring and little fingers all phalanges are mechanically coupled, and a single motor is responsible for the actuation.

4.2 Calibration phase

The purpose of the calibration is to fit a linear model to the encoders of the joints that are mechanically coupled. In order to be as general as possible, we assume that the data are generic n dimensional vectors, here indicated $q \in R^n$. For the thumb, index and middle distal joints $n=2$, while for the ring and little fingers $n=6$ (see footnote on page 17). The parametric linear model used to fit the data is:

$$r: \; q = k_0 + k_1 \cdot t, \qquad t \in [t_{\min}, t_{\max}] \tag{22}$$

where t is the free parameter to be chosen in $[t_{\min}, t_{\max}]$. Another possible representation for this model can be obtained by observing that it represents a line r (i.e. a one dimensional subspace) embedded in a n dimensional vector space. Therefore, it can be represented as the intersection of planes. The general implicit equation of a plane π in R^n is:

$$\pi: \quad a_1 q_1 + a_2 q_2 + ... + a_n q_n = c. \tag{23}$$

Since the plane is a $n-1$ dimensional subspace, the intersection of k planes is a $n-k$ dimensional subspace. Therefore, a line r can be represented by intersecting $n-1$ planes:

$$r: \begin{cases} a_{2,1} q_1 + a_{2,2} q_2 + ... + a_{2,n} q_n = c_2 \\ \vdots \\ a_{n,1} q_1 + a_{n,2} q_2 + ... + a_{n,n} q_n = c_n \end{cases}. \tag{24}$$

However, the representation of a line by means of the coefficients in (24) is redundant. It can be shown that an equivalent non-redundant implicit representation is the following:

$$r: \begin{cases} a_{2,1} \cdot q_1 + q_2 + 0 \cdot q_3 ... + 0 \cdot q_n = c_2 \\ \vdots \\ a_{n,1} \cdot q_1 + 0 \cdot q_2 + ... + 0 \cdot q_{n-1} + q_n = c_n \end{cases}, \tag{25}$$

which corresponds to the following parametric model:

$$r: \; q = k_0 + k_1 \cdot t, \tag{26}$$

with $t = q_1$ and:

$$k_0 = \begin{bmatrix} 0 \\ c_{22} \\ \vdots \\ c_n \end{bmatrix}, \; k_1 = \begin{bmatrix} 1 \\ -a_{2,1} \\ \vdots \\ -a_{n,1} \end{bmatrix}. \tag{27}$$

Given a set $q^1, ..., q^N$ of observations, the parameters $a_{i,1}$ and c_i are estimated by solving the following least squares optimization:

$$\min_{\substack{a_{2,1},...,a_{n,1} \\ c_2,...,c_n}} \sum_{i=1}^{N} \sum_{j=2}^{n} \left\| a_{j,1} \cdot q_1^i + q_j^i - c_j \right\|^2. \tag{28}$$

These parameters are then converted to k_0 and k_1 by means of equation (27). Moreover, points of the minimum distance are computed as follows:

$$t_j^* = \arg\min_t \left\| k_0 + k_1 \cdot t - q^j \right\|, \; j = 1...N, \tag{29}$$

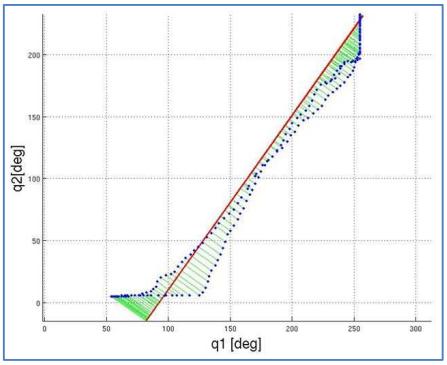

Figure 14. The linear manifold that models the coupled distal joints of the thumb.

which corresponds to:

$$q^{*,j} = k_0 + k_1 \cdot t_j^*, \ j = 1 \ldots N. \tag{30}$$

Accordingly, the span of allowed values for t is determined as follows:

$$t_{\max} = \max_{j=1\ldots N} t_j^*, \ t_{\min} = \min_{j=1\ldots N} t_j^*. \tag{31}$$

Similarly, the maximum and the minimum distance from the model is computed as:

$$q_{\max} = \max_{j=1\ldots N} \left\| q^{*,j} - q^j \right\|, \ q_{\min} = \min_{j=1\ldots N} \left\| q^{*,j} - q^j \right\|. \tag{32}$$

Figure 14 shows the result of an example calibration carried out on the two distal joints of the *iCub* thumb. The two joints are represented by blue dots: clearly, they roughly lie on a linear manifold represented by the red line, whilst the distances from this manifold are depicted with green lines. Therefore by measuring the deviation from the linear manifold and by applying proper thresholds, this method allows to detect contacts with objects during grasping tasks and to safely stop the fingers preventing damage.

5 Impedance Control

The importance of robot safety has been discussed in [7-9]. It has been noted that all the aspects of robot design should be considered to increase safety (mechanics design is at the basis of the new trend of safe robots, but also software, and electronics should be considered). From the mechanical point of view, passive compliance has the peculiarity of decoupling the link and rotor's inertia, thus resulting in an intrinsically safe actuation system [10, 11]. Light-weight designs have also been investigated in [12] where it is shown that impact forces can be reduced, resulting in a safer robot. The macro-mini actuation design has been investigated in [13], where it is shown that relocating the major source of actuation at the base of the manipulator and employing a light-weight and small motor, high-frequency torque capability is maintained, without increasing the impedance of the system.

An alternative to these mechanical solutions is to achieve active force control, or active joint torque control. Even though this approach is far from being intrinsically safe, it does not require to increase the complexity of the mechanical system. Standard industrial approaches employ an F/T sensor located at the end effector of the manipulator. The obvious assumption in this case is that the robot interaction with the environment only occurs at the tool level. Joint torque sensing might be an

alternative solution to this issue [14,16], but it again requires a specific joint design for exploiting torque sensing.

The iCub robot instead is arranged with a solution which embodies both the benefits of F/T and joint torque sensing. It mounts an F/T sensor at the beginning of the kinematic chain of each limb. This makes the robot sensitive also to interaction occurring at different level (not only at the end-effector) and also gives information about the real torque acting on the joint, due to the limb dynamic.

We first report here a brief description of the custom F/T sensors which have been employed for the experiments. In the following sections we illustrate the method used for effective joint torque estimation and the controllers that we have implemented. Finally, we show the experiments conducted for identification, controller regulation and validation.

5.1 The Force Sensor

We employ a 6-axis force/torque sensor, integrated within the two arms and legs [17]. The F/T sensors of the arms are placed in the upper arm, between the shoulder and the elbow, while those employed for the legs are placed between the hip joints and the knee Figure 15. These F/T sensors employ semiconductor strain gages (SSG) for measuring the deformation of the sensing elements.

The electronics that performs signal conditioning and A/D conversion is embedded in the sensor. The board is based on a 16 bit DSP from Microchip (dsPIC30F4013). It samples up to a maximum of 6 analog channels for strain gauges sensors in a bridge configuration (using an instrumentation amplifier INA155); the Analog to digital converter (AD7685, 16 bit, 250 KSps, SPI interface) is multiplexed (ADG658) on the 6 channels.

Commonly force sensors are placed at the end-effector. As already anticipated, on the iCub the F/T sensors are mounted at the beginning of the kinematic chain of each limb (i.e. one F/T sensor for each arm and for each leg). This solution however has some advantages, in particular it allows to:

- estimate forces and torques due to the internal dynamic of the links;
- measure external forces exerted on the whole arm;
- derive the torques at each joint.

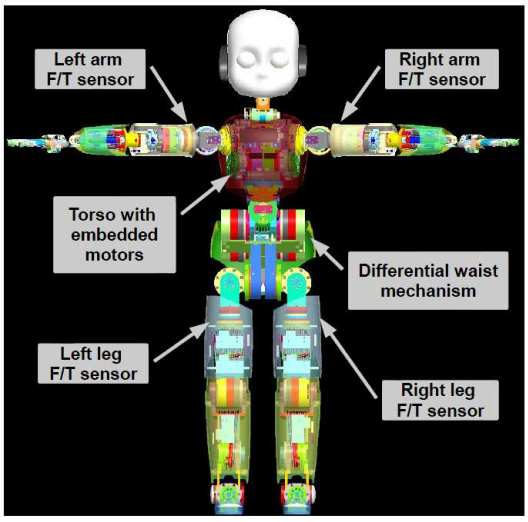

Figure 15. A schematic representation of the iCub that shows the location of the force/torque sensors.

The drawback is that, if not compensated for, dynamic forces due to the links are detected as external forces (something that does not happen when the F/T sensor is placed at the end-effector). To properly compensate forces acting on the whole arm, we need to know their point of application. Since the latter information is not available without tactile feedback in this work we assumed that all forced are applied at the end-effector.

5.2 Force and Impedance Control

Let us assume that the point of application of the external forces is the end-effector of our manipulator. Suppose also that the contribution of the internal dynamics of the manipulator is negligible. In these hypotheses the actual external wrench and the measured F/T vector are related with pure kinematic relations, and it is possible to estimate the corresponding joint level torques $\hat{\tau} \in R^n$.

With reference to Figure 16, the output of the F/T is a wrench ${}^{s}F_s \in R^6$ represented in the sensor reference frame <s>. We can compute an equivalent wrench ${}^{b}F_e \in R^6$ that, applied to the end-effector, produces the same effect. This clearly depends on the vector $p_s^e \in R^3$ – the relative distance of the center of the sensor reference frame $O_s \in R^3$ with respect to the position of the end-effector $O_e \in R^3$ – and on the rotation matrix relating <s> with <e>:

$$ {}^{b}F_e = T_e^b \; H_s^e \; {}^{s}F_s \tag{33} $$

being ${}^{b}F_e \in R^6$ the external force (represented in the reference frame <b>) and H_s^e and $T_e^b \in R^{6x6}$ are defined as:

$$ H_s^e = \begin{bmatrix} R_s^e & 0 \\ -S(p)R_s^e & R_s^e \end{bmatrix}, \tag{34} $$

$$ T_e^b = \begin{bmatrix} R_e^b & 0 \\ 0 & R_e^b \end{bmatrix}. \tag{35} $$

Here $R_a^b \in R^{3x3}$ represent the rotation matrix from <a> to <b>, and $S(.) \in R^{3x3}$ the operator performing the cross product $p\times$.

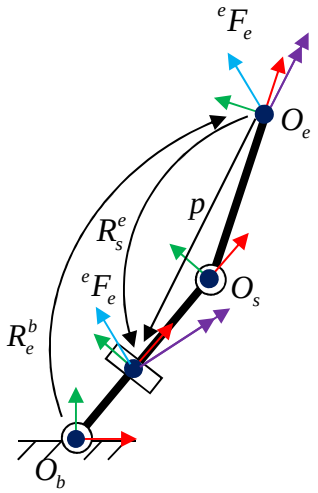

Figure 16. Schematic representation of the F/T sensor, external forces and torque level forces.

From ${}^{b}F_{e}$ it is straightforward to compute the joint level torques:

$$\hat{\tau} = J^{T}(q)\,{}^{b}F_{e}, \tag{36}$$

where $J(q) \in R^{6xn}$ is the Jacobian of the manipulator.

Given the value of $\hat{\tau}$ corresponding to the current reading ${}^{s}F_{s}$, the following control strategy:

$$u = PID(\hat{\tau} - \tau_{d}) \tag{37}$$

computes the motor command $u \in R^{n}$ that achieves a desired value of joint torques $\tau_{d} \in R^{n}$ (which, in turn, produce a corresponding net force exerted by the arm at the end effector). A simple way to demonstrate force control is to realize an impedance controller. At the joint level this is easily achieved by computing τ_{d} as in:

$$\tau_{d} = -K(q - q*) \tag{38}$$

being $K \in R^{n}$ the vector of virtual joint stiffnesses. This controller simulates virtual springs attached to each joint, with stiffness K_{i} and equilibrium point at q_{i}^{*}.

In theory if K is *large enough* $q \to q*$ and we can use this controller to achieve any desired configuration of the arm. In practice, however, we would like to use small values of the stiffness *K* so to reduce the effects of unwanted collisions. In this case we need to model the dynamic forces produced by the arm.

5.3 Experiments

We tested the closed loop response of the system in two conditions: as a pure torque controller and as an impedance controller.

In the first experiment we measured the step response of the torque controller described in (37). We control the elbow joint by varying τ_{d} in steps of +/-1.5 Nm. Figure 17 shows the response of the system whereas Figure 18 reports the error of the controller. The response of the system is oscillatory but has fast convergence to the steady value. The steady state error is due to the fact that the controller is purely proportional (Kd and Ki were set to zero).

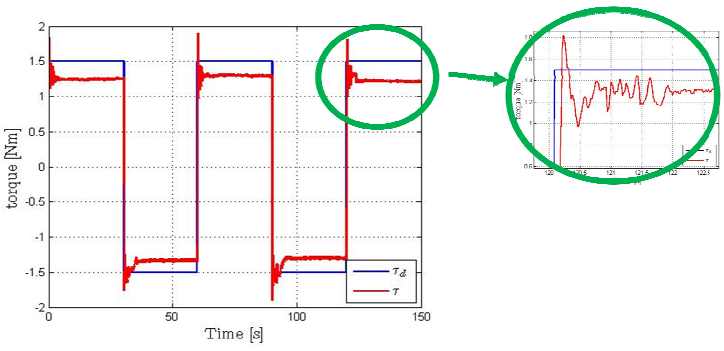

Figure 17. Torque controller step response, elbow joint.

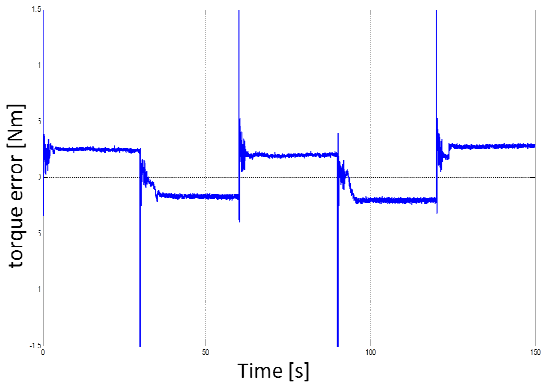

Figure 18. Controller error, elbow joint.

In the second experiment we tested the performance of the impedance controller described in (38). The controller now maintains $q*$, or equivalently, a certain position of the arm. We apply disturbances by means of variable forces applied at the end-effector. Forces produce a displacement of the end-effector; depending on the stiffness K the controller tries to oppose the external disturbances with a restoring force proportional to the displacement. This situation is similar to what happens when the arm interacts with the environment.

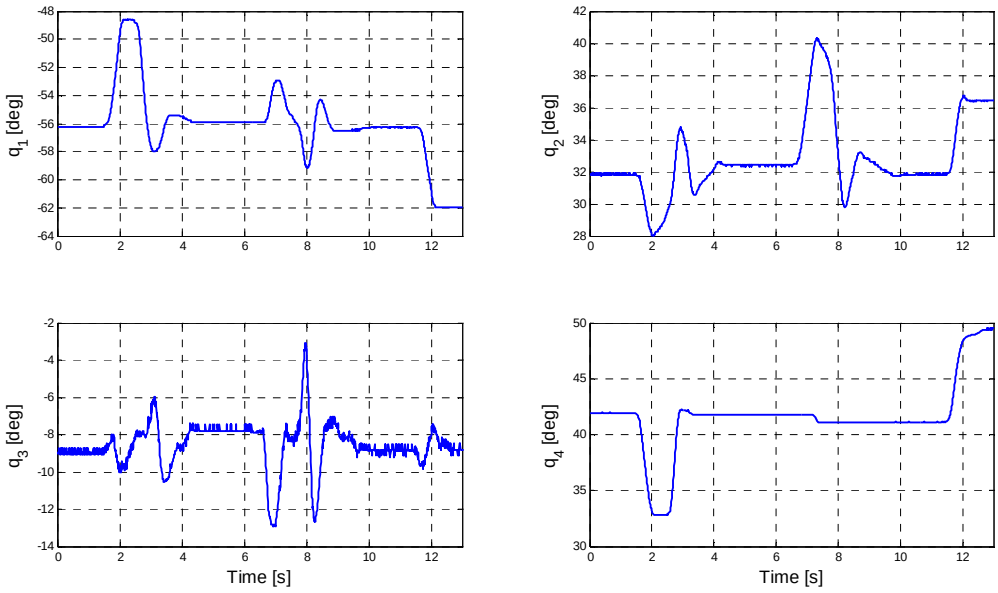

Figure 19. Impedance controller: displacement of each joint during interaction/external disturbances. Cyan dashed lines represent when the interaction occurs. The blue solid line shows the encoder value. During the whole experiment the reference value requested to the controller is maintained stationary $q* = const$.

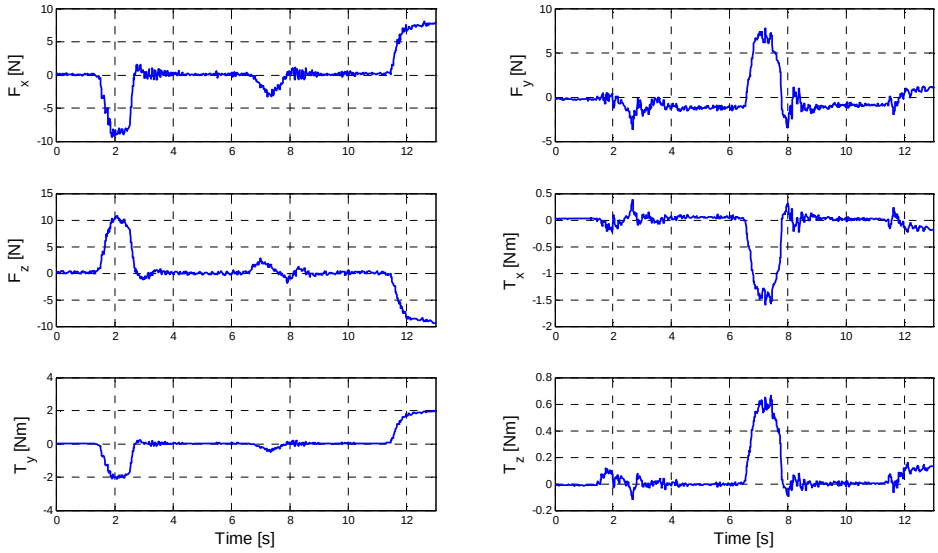

Figure 20. Force sensed by the F/T sensor during the same experiment in Figure 19.

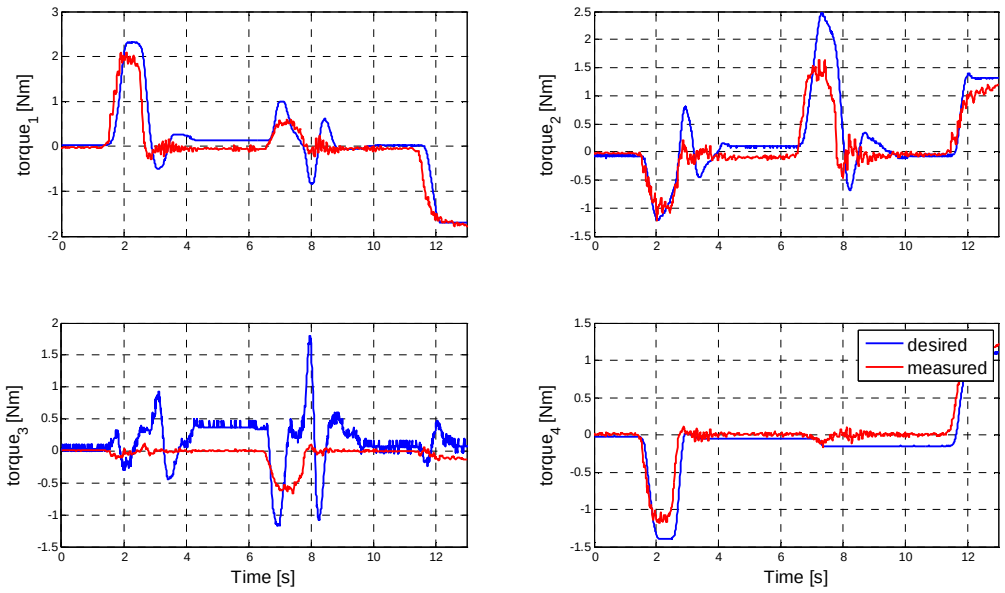

Figure 21. Desired (blue line) versus actual (red line) torques at the joints, during the same experiment in Figure 19.

6 Bibliography

[1] Wächter, L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. Mathematical Programming 106 (1): pp 25–57, 2006.

[2] D. Bullock, S. Grossberg, Neural dynamics of planned arm movements: Emergent invariants and speed-accuracy properties during trajectory formation. Psychological Review, 95(1), pp 49–90, 1988.

[3] M. Hersch, A. G. Billard, Reaching with multi-referential dynamical systems. Autonomous Robots, Springer-Verlag, pp 71–83, 2008.

[4] W. Abend, E. Bizzi, P. Morasso, Human arm trajectory formation. Brain, 105, pp 331–348, 1982.

[5] T. Flash, N. Hogan, The coordination of arm movements: an experimentally confirmed mathematical model. Neuroscience 5: pp 1688–1703, 1985.

[6] B. Hoff, M. A. Arbib, A model of the effects of speed, accuracy and perturbation on visually guided reaching. In: Control of arm movement in space: neurophysiological and computational approaches (R. Caminiti, P. B. Johnson, Y. Burnod, eds), pp 285–306.

[7] A. Desantis, B. Siciliano, A. Deluca, and A. Bicchi, An atlas of physical human robot interaction, Mechanism and Machine Theory, vol. 43, no. 3, pp. 253–270, March 2008.

[8] R. Schiavi, B. A., and F. Flacco, Integration of active and passive compliance control for safe human-robot coexistence, in 2009 IEEE International Conference on Robotics and Automation, Kobe International Conference Center, Kobe, Japan, 12-17 May, 2009.

[9] M. Zinn, O. Khatib, B. Roth, and S. J. K., Playing it safe – human friendly robots, Robotics & Automation Magazine, IEEE, vol. 11, no. 2, pp. 12–21, 2004.

[10] G. Pratt and M. Williamson, Series elastic actuators, in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., vol. 1, Pittsburgh, PA, p. 399-406, 1995.

[11] R. Schiavi, G. Grioli, S. Sen, and B. A., Vsa-II: a novel prototype of variable stiffness actuator for safe and performing robots interacting with humans, in 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 19-23 May 2008.

[12] S. Haddadin, A. A. Albu-Schaffer, and H. G., Safety evaluation of phisical human-robot interaction via crash-testing, in Robotics: Science and System Conference, Atlanta, Georgia, 2007.

[13] B. R. M. Zinn, O. Khatib and J. Salisbury, A new actuation approach for human friendly robot design, in VIII, Springer Tracts in Advanced Robotics, B. Siciliano and E. P. Dario, Eds., Berlin: Spinger-Verlag, 2002.

[14] S. Haddadin, A. Albu-Schaffer, A., and G. Hirzinger, Safety evaluation of phisical human-robot interaction via crash-testing, in Robotics: Science and System Conference (RSS 2007), Atlanta, Georgia, 2007.

[15] A. Parmiggiani, M. Randazzo, L. Natale, G. Metta, and G. Sandini, Joint torque sensing for the upper-body of the icub humanoid robot, in International Conference on Humanoid Robots, Paris, France, 2009.

[16] N. Tsagarakis, F. Becchi, L. Righetti, A. Ijspeert, and D. Caldwell, Lower body realization of the baby humanoid - icub, in IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, USA, Oct-Nov 2007.

[17] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, “The icub humanoid robot: an open platform for research in embodied cognition,” in PerMIS: Performance Metrics for Intelligent Systems Workshop, Washington DC, USA, Aug 19-21, 2008.

[18] M. Fumagalli, A. Gijsberts, S. Ivaldi, L. Jamone, G. Metta, L. Natale, F. Nori, and G. Sandini, “Learning to Exploit Proximal Force Sensing: a Comparison Approach”, from motor learning to interaction learning in robots ed., ser. Studies in Computational Intelligence. Springer- Verlag Berlin and Heidelberg GmbH & Co. K, 2010, vol. Volume 264, pp. 149–167, 2010.