

Force control and reaching movements on the iCub humanoid robot

Giorgio Metta and Lorenzo Natale and Francesco Nori and Giulio Sandini

Abstract This paper is about a layered controller for a complex humanoid robot: namely, the iCub. We exploited a combination of precomputed models and machine learning owing to the principle of balancing the design effort with the complexity of data collection for learning. A first layer uses the iCub sensors to implement impedance control, on top of which we plan trajectories to reach for visually identified targets while avoiding the most obvious joint limits or self collision of the robot arm & body. Modeling errors or parameter misestimation are compensated by machine learning in order to obtain accurate pointing and reaching movement. Motion segmentation is the main visual cue employed by the robot.

1 Introduction

In this paper we consider a solution to the problem of reaching for a visually identified target in a complex humanoid robot platform, considering both potential forceful interactions with objects or people and gross mistakes due to miscalibration of the controller parameters. Our reference platform is the iCub [1], a humanoid robot

Giorgio Metta
Robotics, Brain and Cognitive Sciences, Istituto Italiano di Tecnologia, Via Morego, 30 - Genoa, Italy, e-mail: giorgio.metta@iit.it

Lorenzo Natale
Robotics, Brain and Cognitive Sciences, Istituto Italiano di Tecnologia, Via Morego, 30 - Genoa, Italy, e-mail: lorenzo.natale@iit.it

Francesco Nori
Robotics, Brain and Cognitive Sciences, Istituto Italiano di Tecnologia, Via Morego, 30 - Genoa, Italy, e-mail: francesco.nori@iit.it

Giulio Sandini
Robotics, Brain and Cognitive Sciences, Istituto Italiano di Tecnologia, Via Morego, 30 - Genoa, Italy, e-mail: giulio.sandini@iit.it

shaped as a three and half years old child. The iCub, by design, only uses “passive” sensors as for example cameras, gyroscopes, pressure, force and contact sensors, microphones and so forth. We excluded the use of lasers, sonars and other esoteric sensing modalities.

In this conditions and in an unstructured environment where human can freely move and work (our laboratory space in the daily use of the iCub), it is unlikely that the robot obtains an accurate model of the environment for accurate impact-free planning of movements. One common solution [2] is to control the robot mechanical impedance and, simultaneously, minimize impacts by using for example vision and trajectory planning. The possibility of impedance control lowers the requirements of vision and guarantees a certain degree of safety in case of contacts with the environment – though, strictly speaking, the robot can still be dangerous and cause damage if it moves fast.

The control architecture described in this paper is not very different in principle from a standard computed torque approach [3]. A first layer compensates for the dynamics and linearizes the system. Because of the communication bus of the iCub controllers, of bandwidth requirements, and implementation constraints, it operates in joint space. A second layer subsequently plans trajectories starting from a description of the target position in extrinsic space and merging joint limits, a secondary task specification, inverse kinematics and singularity avoidance. We show in the remainder of the paper how this is implemented by mixing hand-coded models of the robot dynamics and kinematics together with machine learning.

Reaching and pointing is fundamental in learning about the environment enabling interaction with objects and their manipulation to achieve complex tasks. In this sense these are the basic building blocks of a complex cognitive architecture for the iCub.

2 Experimental platform: the iCub

The iCub is one of the results of the RobotCub project, an EU-funded endeavor to create a common platform for researchers interested in embodied artificial cognitive systems [4].

The initial specifications of the robot aimed at replicating the size of a three years old child. In particular, it was required that the robot be capable of crawling on all fours and possess fine manipulation abilities. For a motivation of why these features are important, the interested reader is referred to Metta et al. [5].

Dimensions, kinematic layout and ranges of movement were drafted by considering biomechanical models and anthropometric tables [6]. Rigid body simulations were used to determine the crucial kinematic features in order to perform the set of desired tasks and motions, i.e. reaching, crawling, etc. [7]. These simulations also provided joint torques requirements. Data were then used as a baseline performance indicator for the selection of the actuators. The final kinematic structure of the robot is shown in figure 1c. The iCub has 53 degrees of freedom (DoF). Its kinematics

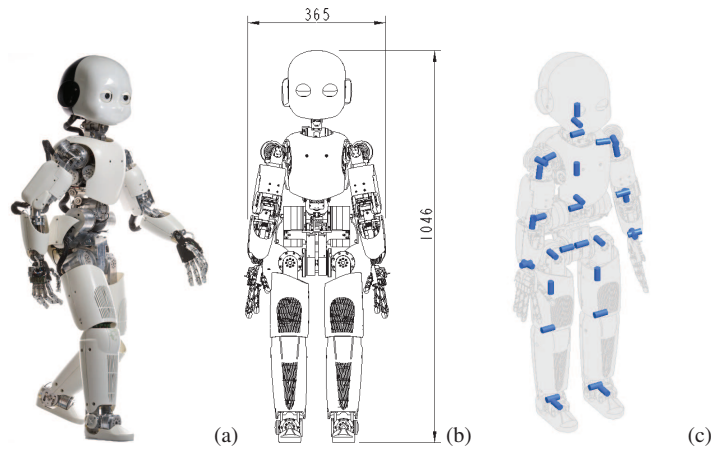


Fig. 1 The iCub platform: panel (a) a picture of the latest realization of the iCub; panel (b) approximate dimensions height \times width; and panel (c) the kinematic structure of the major joints.

has several special features which are rarely found in other humanoid robots: e.g. the waist has three DoF which considerably increase the robot's mobility; the three DoF shoulder joint is constructed to have its axes of rotation always intersecting at one point.

To match the torque requirements we employed rotary electric motors coupled with speed reducers. We found this to be the most suitable choice in terms of robustness and reliability. Motor groups with various characteristics were developed (e.g. 40Nm, 20Nm and 11Nm) for different placements into the iCub. We used the Kollmorgen-DanaherMotion RBE type brushless frameless motor (BLM) and a CSD frameless Harmonic Drive as speed reducer. The use of frameless components allowed further optimization of space and reduced weight. Smaller motors for moving the fingers, eyes and neck are from Fulhaber in various sizes and reduction gear ratios.

Cable drives were used almost everywhere on the iCub. Most joints have relocated motors as for example in the hand, shoulder (besides one joint), elbow, waist and legs (apart from two joints). Cable drives are efficient and almost mandatory in order to optimize the motor locations and the overall "shape" of the robot. All joints in the hand are cable driven. The hand of the iCub has 20 joints which are moved by only 9 motors: this implies that some of the joints are under-actuated and their movement is obtained by means of the cable couplings. Similarly to the human body most of the hand actuation is in the forearm subsection. The head is another particular component of the iCub enabling independent vergence movements supported by a three DoF neck for a total of six DoF.

By design we decided to only use "passive sensors" and in particular cameras, microphones, gyroscopes and accelerometers, force/torque (FTS) and tactile sensors as well as the traditional motor encoders. Of special relevance is the sensorized skin which is not easily found in other platforms as well as the force/torque sensors that

are used for force/impedance control (see later). No active sensing is provided as for example lasers, structured light projectors, and so forth.

The iCub mounts custom-designed electronics which consists of programmable controller cards, amplifiers, DACs and digital I/O cards. This ecosystem of micro-controller cards relies on multiple CAN bus lines (up to 10) for communication and synchronization and then connects with a cluster of external machines via a Gbit/s Ethernet network. Data are acquired and synchronized (and timestamped) before being made available on the network. We designed the software middleware that supports data acquisition and control of the robot as well as all the firmware that operates on the microcontrollers which eventually drive each single transistor that moves the motors.

The software middleware is called YARP [8]. YARP is a thin library that enables multi-platform and multi-IDE development and collaboration by providing a layer that shields the users from the quirks of the underlying operating system and robot control. The complete design of the iCub (drawings, schematics, specifications) and its software (both middleware and controllers) is distributed according to the GPL or the LGPL licenses.

3 Dynamics

The first layer of the proposed architecture is based on computation of the body dynamics and implements joint position & velocity control on top of joint-level impedance. In the simplest possible version, the controller cards implement a 1ms feedback loop relying on the error e defined as:

$$e = \tau - \tau_d, \quad (1)$$

where τ is the vector of joint torques and τ_d its desired value. We do not know τ directly on the iCub but we have access to estimates through the force/torque sensors (FTSs). They are mounted as indicated in figure 2 in the upper part of the limbs and can therefore be used to detect wrenches at any location in the iCub limbs and not only at the end-effector as it is more typical for industrial manipulators.

We show that τ can be estimated from the FTS measurements of each given limb (equations repeat identical on each limb). Let's indicate with w_s the wrench measured by the FTS and assume that it is due to an actual external wrench at a known location (e.g. at the end-effector) which we call w_e . We can estimate w_e by propagating the measurement on the kinematic chain of the limb (changing coordinates):

$$\hat{w}_e = \begin{bmatrix} I & 0 \\ -[\bar{r}_{se}]_{\times} & I \end{bmatrix} \cdot (w_s - w_i), \quad (2)$$

with $[\bar{r}_{se}]_{\times}$ the skew-symmetric matrix representing the cross product with the vector \bar{r}_{se} , \hat{w}_e the estimate of w_e , and w_i the internal wrench (due to internal forces and moments). Note that $[\bar{r}_{se}]_{\times}$ is a function of q , the vector of joint angles. w_i can be

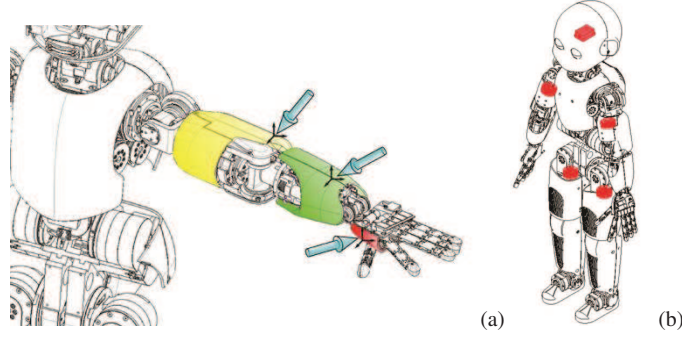


Fig. 2 In (a) a typical interaction of the iCub arm with the environment exemplified here with a number of wrenches at different locations and in (b) the location of the four FTSS of the iCub in the upper part of the limbs (proximal with respect to the reference frame of the robot kinematic chains) and of the inertial sensors mounted in the head.

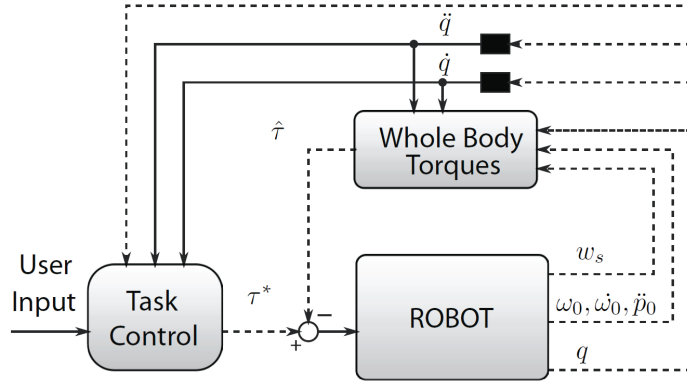


Fig. 3 The torque controller of the iCub. See text for details.

estimated from the dynamics of the limb (either with the Lagrange or Newton-Euler formulation). To estimate τ_e we only need to project \hat{w}_e to the joint torques using the transposed Jacobian, i.e.:

$$\hat{\tau}_e = J^T(q) \cdot \hat{w}_e . \quad (3)$$

We can then use this estimate in a control loop by defining the torque error e as:

$$e = \hat{\tau}_e - \tau_d , \quad (4)$$

where $\hat{\tau}_e$ is an estimate of τ regulated by a PID controller of the form:

$$u = k_p \cdot e + k_d \cdot \dot{e} + k_i \cdot \int e , \quad (5)$$

where k_p , k_d and k_i are the usual PID gains and u the amplifier output (the PWM duty cycle which determines the equivalent applied voltage at the motor). Similarly we can build an impedance controller in joint space by making τ_d of the form:

$$\tau_d = K \cdot (q - q_d) + D \cdot (\dot{q} - \dot{q}_d), \quad (6)$$

which can be implemented at the controller card level if K and D are diagonal matrices. Furthermore, we can command velocity by making:

$$\dot{q}_d(t) = \dot{q}_d(t - \delta t) + \dot{q}_d(t) \delta t, \quad (7)$$

with δt the control cycle interval (1ms in our case). This latter modality is useful when generating whole trajectories incrementally. The actual computation of the dynamics and kinematics is based on a graph representation which we detail in the following.

We start by considering an open (single or multiple branches) kinematic chain with n DoF composed of $n + 1$ links. Adopting the Denavit-Hartenberg notation [3], we define a set of reference frames $\langle 0 \rangle, \langle 1 \rangle, \dots, \langle n \rangle$, attached at each link. The i^{th} link of the chain is described by a vertex v_i (sometimes called node), usually represented by the symbol \textcircled{i} . A hinge joint between the link i and the link j (i.e. a rotational joint) is represented by an oriented edge $e_{i,j}$ connecting v_i with v_j : $\textcircled{i} \rightarrow \textcircled{j}$. In a n DoF open chain, each vertex (except for the initial and terminal, v_0 and v_n respectively) has two edges. Therefore, the graph representation of the n -link chain is an oriented sequence of nodes v_i , connected by edges $e_{i-1,i}$. The orientation of the edges can be either chosen arbitrarily (it will be clear later on that the orientation simply induces a convention) or it can follow from the exploration of the kinematic tree according to the *regular numbering scheme* [9], which induces a parent-child relationship such that each node has a unique input edge and multiple output edges. We further follow the classical Denavit-Hartenberg notation, we assume that each joint has an associated reference frame with the z -axis aligned with the rotation axis; this frame will be denoted $\langle e_{i,j} \rangle$. In kinematics, an edge $e_{i,j}$ from v_i to v_j represents the fact that $\langle e_{i,j} \rangle$ is fixed in the i^{th} link. In dynamics, $e_{i,j}$ represents the fact that the dynamic equations will compute (and make use of) $w_{i,j}$, i.e. the wrench that the i^{th} link exerts on the j^{th} link, and not the equal and opposite reaction $-w_{i,j}$, i.e. the wrench that the j^{th} link exerts on the i^{th} link. In order to simplify the computations of the inverse dynamics on the graph, kinematic and dynamic measurements have been explicitly represented. Specifically, the graph representation has been enhanced with a new set of graphical symbols: a triangle to represent kinematic quantities (i.e. velocities and acceleration of links - $\omega, \dot{\omega}, \dot{p}, \ddot{p}$), and a rhombus for wrenches (i.e. force sensors measurements on a link - f, μ). Moreover these symbols have been further divided into known quantities to represent sensors measurements, and unknown to indicate the quantities to be computed, as in the following:

- ∇ : unknown kinematic information
- \blacktriangledown : known (e.g., measured) kinematic information
- \blacklozenge : unknown dynamic information

- \blacklozenge : known (e.g., measured) dynamic information

In general, kinematic variables can be measured by means of gyroscopes, accelerometers, or simply inertial sensors. When attached on link i^{th} , these sensors provide angular and linear velocities and accelerations (ω , $\dot{\omega}$, \dot{p} and \ddot{p}) at the specific location where the sensor is located. We can represent these measurement in the graph with a *black triangle* (\blacktriangledown) and an additional edge from the proper link where the sensor is attached to the triangle. As usual, the edge has an associated reference frame, in this case corresponding to the reference frame of the sensor. An unknown kinematic variable is represented by a *white triangle* (∇) with an associated edge going from the link (where the unknown kinematic variable is attached) to the triangle. Similarly, we introduce two new types of nodes with a rhomboidal shape: *black rhombus* (\blacklozenge) to represent known (i.e. measured) wrenches, *white rhombus* (\lozenge) to represent unknown wrenches which need to be computed. The reference frame associated to the edge will be the location of the applied or unknown wrench. The complete graph for the iCub is shown in figure 4.

From the graph structure, we can define the update rule that brings information across edges and by traversing the graph we therefore compute either dynamical or kinematic unknowns (\lozenge and ∇ respectively). For kinematic quantities this is:

$$\begin{aligned}\omega_{i+1} &= \omega_i + \dot{\theta}_{i+1} z_i, \\ \dot{\omega}_{i+1} &= \dot{\omega}_i + \ddot{\theta}_{i+1} z_i + \dot{\theta}_{i+1} \omega_i \times z_i, \\ \ddot{p}_{i+1} &= \ddot{p}_i + \dot{\omega}_i \times r_{i,i+1} + \omega_{i+1} \times (\omega_{i+1} \times r_{i,i+1}),\end{aligned}\quad (8)$$

where z_i is the z -axis of (i) , i.e. we propagate information from the base to the end-effector visiting all nodes and moving from one node to the next following the edges. The internal dynamics of the manipulator can be studied as well: if the dynamical parameters of the system are known (mass m_i , inertia I_i , center of mass C_i), then we can propagate knowledge of wrenches applied to e.g. the end-effector (f_{n+1} and μ_{n+1}) to the base frame of the manipulator so as to retrieve forces and moments f_i , μ_i :

$$\begin{aligned}f_i &= f_{i+1} + m_i \ddot{p}_{C_i}, \\ \mu_i &= \mu_{i+1} - f_i \times r_{i-1,C_i} + f_{i+1} \times r_{i,C_i} + I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i),\end{aligned}\quad (9)$$

where:

$$\ddot{p}_{C_i} = \ddot{p}_i + \dot{\omega}_i \times r_{i,C_i} + \omega_i \times (\omega_i \times r_{i,C_i}), \quad (10)$$

noting that these are the classical recursive Newton-Euler equations. Knowledge of wrenches enables the computation of w_i as needed in equation 2 or the corresponding joint torques from $\tau_i = \mu_i^T z_{i-1}$.

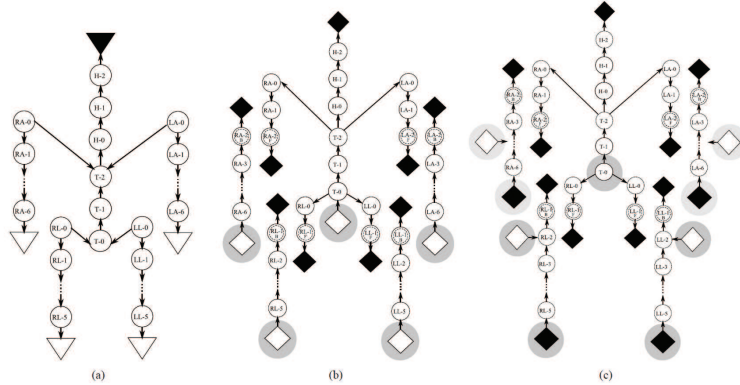


Fig. 4 Representation of iCub’s kinematic and dynamic graph. In (a): iCub’s kinematics. The inertial sensor measure (\blacktriangledown) is the unique source of kinematic information for the whole branched system. (b): iCub’s dynamics when the robot is standing on the mainstay and moving freely in space. Given the four FTSS, the main graph is cut by the four links hosting the sensors, and a total of five sub-graphs are finally generated. The unknowns are the external wrenches at the end-effectors: if the robot does not collide with the environment, they are zero, whereas if a collision happens, then an external wrench arises. The displacement between the expected and the estimated wrenches allows detecting contacts with the environment under the hypothesis that interactions can only occur at the end-effectors. The external wrench on top of the head is assumed to be null. Notice that the mainstay is represented by a unknown wrench \diamond . (c): iCub’s dynamics when the robot is crawling (four points of contact with the ground). As in the previous case, five sub-graphs are generated after the insertion of the four FTSS measurements, but unlike the free-standing case, here the mainstay wrench is removed, being the iCub on the floor. Specific locations for the contacts with the environment are given as part of the task: the unknown external wrenches (\diamond) are placed at wrists and knees, while wrenches at the feet and palms are assumed known and null (\blacktriangledown). Interestingly, while moving on the floor the contact with the upper part could be varying (e.g. wrists, palms, elbows), so the unknown wrenches could be placed in different locations than the ones shown in the graph.

3.1 Validation and further improvements

In order to validate computation of dynamics, we compared measurements from the FTSS with their model-based prediction. The wrenches w_s from the four six-axes FTSS embedded in the limbs are compared with the analogous quantities \hat{w}_s predicted by the dynamical model, during unconstrained movements (i.e. null external wrenches). Kinematic and dynamic parameters are retrieved from the CAD model of the robot. Sensor measurements w_s can be predicted assuming known wrenches at the limbs extremities (hands or feet) and then propagating forces up to the sensors. In this case, null wrenches are assumed, because of the absence of contact with the environment. Table 1 summarizes the statistics of the errors ($w_s - \hat{w}_s$) for each limb during a given, periodic sequence of movements, with the robot supported by a rigid metallic mainstay, and with the limbs moving freely without self collision or contact with the environment. Table 1 shows the mean and the standard devia-

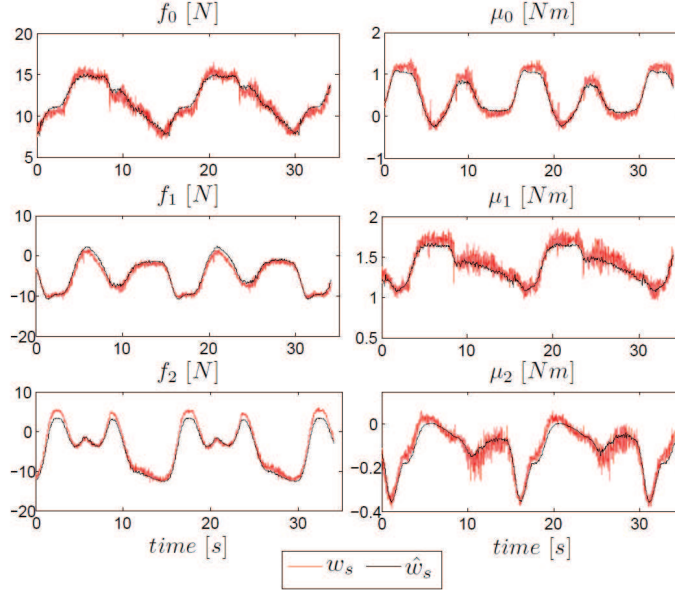


Fig. 5 Comparison between the wrench measured by the FT sensor and that predicted by the model, during a generic contact-free movement of the left arm. The three plots on the left are forces expressed in $[N]$; the three rightmost plots are the moments in $[Nm]$.

tion of the errors between measured and predicted sensor wrench during movement. Figure 5 shows a comparison between w_s and \hat{w}_s for the left arm (without loss of generality).

Subsequently we investigated methods to improve the estimates of the robot dynamics. In another set of experiments we thus compared various non-parametric learning methods with the rigid body model just presented. We refer the interested reader to Gijsberts et al. [10]. We report here only the main findings. The task of learning here is the estimation of the wrenches due to the internal dynamics (w_i) given the FTS readings (w_s) and the robot configuration (q, \dot{q}, \ddot{q}); we do not take into account inertial information.

We compared various methods from the literature as for example the widely used Local Weighted Projection Regression (LWPR), the Local Gaussian Process (LGP) and Gaussian Process Regression (GPR) as presented by Nguyen-Tuong et al. [11] with an incremental version of Kernel Ridge Regression (also known as Sparse Spectrum Gaussian Process) with the aim of maintaining eventually an incremental open-ended learner updating the estimation of the robot dynamics on-line. Our incremental method relies on an approximation of the kernel (see [12]) based on a random sampling of its Fourier spectrum. The more random features, the better the approximation. We considered approximations with 500, 1000, and 2000 features. In the following we call KRR the plain kernel ridge regression method and RFRR^D

Table 1 Error in predicting FT sensor measurement (see text for details).

	ϵ_{f0}	ϵ_{f1}	ϵ_{f2}	$\epsilon_{\mu0}$	$\epsilon_{\mu1}$	$\epsilon_{\mu2}$
$\bar{\epsilon}$	-0.3157	-0.5209	0.7723	-0.0252	0.0582	0.0197
σ_{ϵ}	0.5845	0.7156	0.7550	0.0882	0.0688	0.0364
right arm: $\epsilon \equiv \hat{w}_{s,RA} - w_{s,RA}$						
$\bar{\epsilon}$	-0.0908	-0.4811	0.8699	0.0436	0.0382	0.0030
σ_{ϵ}	0.5742	0.6677	0.7920	0.1048	0.0702	0.0332
left arm: $\epsilon \equiv \hat{w}_{s,LA} - w_{s,LA}$						
$\bar{\epsilon}$	-1.6678	3.4476	-1.5505	0.4050	-0.7340	0.0171
σ_{ϵ}	3.3146	2.7039	1.7996	0.3423	0.7141	0.0771
right leg: $\epsilon \equiv \hat{w}_{s,RL} - w_{s,RL}$						
$\bar{\epsilon}$	0.2941	-5.1476	-1.9459	-0.3084	-0.8399	0.0270
σ_{ϵ}	1.8031	1.8327	2.3490	0.3365	0.8348	0.0498
left leg: $\epsilon \equiv \hat{w}_{s,LL} - w_{s,LL}$						

SI units: $f : [N]$, $\mu : [Nm]$

the random feature version for D features. Various datasets (e.g. Barret, Sarcos) were used from the literature (for comparison [11]) before applying the method to the iCub.

The results in figure 6 show that KRR often outperforms GPR by a significant margin, even though both methods have identical formulations for the predictive mean and KRR hyperparameters were optimized using GPR. These deviations indicate that different hyperparameter configurations were used in both experiments. This is a common problem with GPR in comparative studies: the marginal likelihood is non-convex and its optimization often results in a local optimum that depends on the initial configuration. Hence, we have to be cautious when interpreting the comparative results on these datasets with respect to generalization performance. The comparison between KRR and RFRR, trained using identical hyperparameters, remains valid and gives an indication of the approximation quality of RFRR. As expected, the performance of RFRR steadily improves as the number of random features increases. Furthermore, RFRR^{1000} is often sufficient to obtain satisfactory predictions on all datasets. RFRR^{500} , on the other hand, performs poorly on the Barrett dataset, despite using distinct hyperparameter configurations for each degree of freedom. In this case, RFRR^{1000} with a shared hyperparameter configuration is more accurate and requires overall less time for prediction.

Figure 7 shows how the average nMSE develops as test samples are predicted in sequential order using either KRR or RFRR. RFRR requires between 5000 and 10000 samples to achieve performance comparable to KRR. The performance of KRR, on the other hand, decreases over time. In particular on the iCub dataset it suffers a number of large errors, causing the average nMSE to show sudden jumps. This is a direct consequence of the unavoidable fact that training and test samples are

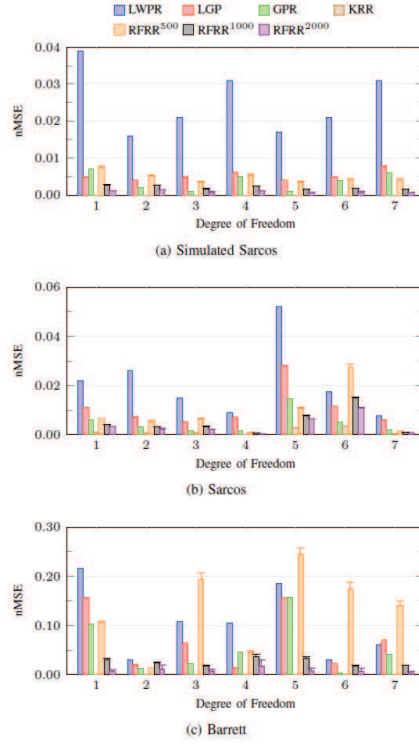


Fig. 6 Prediction error per degree of freedom for the (a) Simulated Sarcos, (b) Sarcos, and (c) Barrett datasets. The results for LWPR, GPR, and LGP are taken from Nguyen-Tuong et al. [11]. The mean error over 25 runs is reported for RFRR with $D \in \{500, 1000, 2000\}$, whereas error bars mark a distance of one standard deviation. Note that in some cases the prediction errors for KRR are very close to zero and therefore barely noticeable.

not guaranteed to be drawn from the same distribution. Incremental RFRR, on the other hand, is largely unaffected by these changes and demonstrates stable predictive performance. This is not surprising, as RFRR is incremental and thus (1) it is able to adapt to changing conditions, and (2) it eventually has trained on significantly more samples than KRR. Furthermore, figure 7 shows that 200 random features are sufficient to achieve satisfactory performance on either dataset. In this case, model updates of RFRR require only $400\mu s$, as compared to $2ms$ and $7ms$ when using 500 or 1000 random features, respectively. These timing figures make incremental RFRR suitable for high frequency loops as needed in robot control tasks.

In conclusion, this shows that for a relatively complex robot like the iCub, good estimation of the internal dynamics is possible and that a combination of non-parametric and parametric methods can provide simultaneously good generalization performance, fast and incremental learning. Not surprisingly, lower errors are

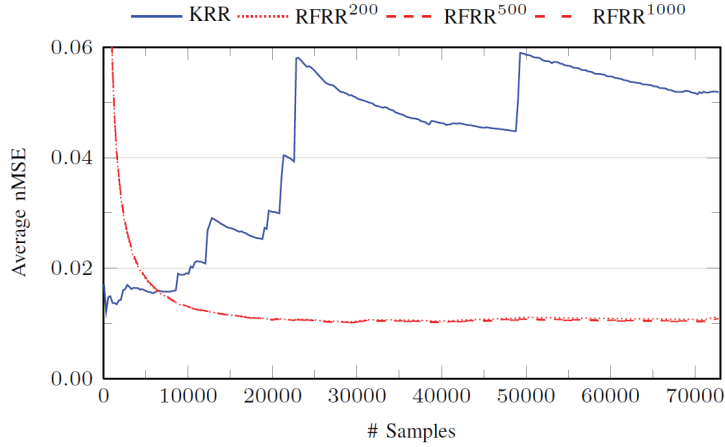


Fig. 7 Average prediction error with respect to the number of test samples of KRR and incremental RFRR with $D \in \{200, 500, 1000\}$ on the iCub dataset. The error is measured as the nMSE averaged over the force and torque output components. The standard deviation over 25 runs of RFRR is negligible in all cases, for clarity we report only the mean without error bars.

obtained with learning. In the next section we see how to build on this controller to reach for visually identified targets.

4 Kinematics

We consider the general problem of computing the value of joint angles q_d in order to reach a given position in space $x_d \in \mathbb{R}^3$ and orientation $\alpha_d \in \mathbb{R}^4$ of the end-effector (where α_d is a representation of rotation in axis/angle notation). Note that q_d can be directly connected to the input of the impedance controller described in section 3. It is desired that the computed solution satisfies a set of additional constraints expressed as generic inequalities – we see later the reason for constraining the solution. This can be stated as follows:

$$q_d = \operatorname{argmin}_{q \in \mathbb{R}^n} (\|\alpha_d - K_\alpha(q)\|^2 + \beta(q_{rest} - q)^T W (q_{rest} - q)), \quad (11)$$

$$s.t. \begin{cases} \|x_d - K_x(q)\|^2 < \varepsilon \\ q_L < q < q_U \end{cases},$$

where K_x and K_α are the forward kinematic functions for the position and orientation of the end-effector for a given configuration q ; q_{rest} is a preferred joint configuration, W is a diagonal weighting matrix, β a positive scalar weighting the influence of the terms in the optimization and ε a parameter for tuning the precision of the movement. Typically $\beta < 1$ and $\varepsilon \in [10^{-5}, 10^{-4}]$. The solution to equation 11 has to satisfy the set of additional constraints of joint limits $q_L < q < q_U$ with q_L, q_U the

lower and upper bounds respectively. In the case of the iCub, we solved this problem for ten DoF – seven of the arm and three of the waist and we determined the value of q_{rest} so that the waist is as upright as possible. The left and right arm can be both controlled by switching from one or the other kinematic chain (e.g. as a function of the distance to the target).

We used an interior point optimization technique to solve the problem in 11. In particular we used IpOpt [13], a public domain software package designed for large-scale nonlinear optimization. This approach has the following advantages:

1. Quick convergence. IpOpt is reliable and fast enough to be employed in control loops at reasonable rates (tens of milliseconds), as e.g. compared to more traditional iterative methods such as the Cyclic Coordinate Descent (CCD) adopted in [14];
2. Scalability. The intrinsic capability of the optimizer to treat nonlinear problems in any arbitrary number of variables is exploited to make the controller structure easily scalable with the size of the joint space. For example, it is possible to change at run time from the control of the 7-DoF iCub arm to the complete 10-DoF structure inclusive of the waist or to any combination of the joints depending on the task;
3. Automatic handling of singularities and joint limits. This technique automatically deals with singularities in the arm Jacobian and joint limits, and can find solutions in virtually any working conditions;
4. Tasks hierarchy. The task is split in two subtasks: the control of the orientation and the control of the position of the end-effector. Different priorities can be assigned to the subtasks. In our case the control of position has higher priority with respect to orientation (the former is handled as a nonlinear constraint and thus is evaluated before the cost);
5. 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 instance, we added a set of constraints that avoid reaching the limits of the tendons that actuate the three joints of the shoulder.

Once q_d is determined as described above, there is still the problem of generating a trajectory from the current robot configuration q to q_d . Simultaneously, we would like to impose suitable smoothness constraints to the trajectory. This has been obtained by using the Multi-Referential Dynamical Systems approach [14], whereby two dynamical controllers, one in joint space and another in task space, evolve concurrently (figure 8). The coherence constraint, that is $\dot{x} = J\dot{q}$, with J the Jacobian of the kinematics map, guarantees that at each instant of time the trajectory is meaningful. This is enforced by using the Lagrangian multipliers method and can be tuned to modulate the relative influence of each controller (i.e. to avoid joint angles limits). The advantage of such a redundant representation includes the management of the singularities while maintaining a quasi-straight trajectory profile of the end-effector in the task space – reproducing a human-like behavior [15].

Differently from the work of Hersch and Billard, we designed a feedback trajectory generator instead of the VITE (Vector- Integration-To-Endpoint) method used

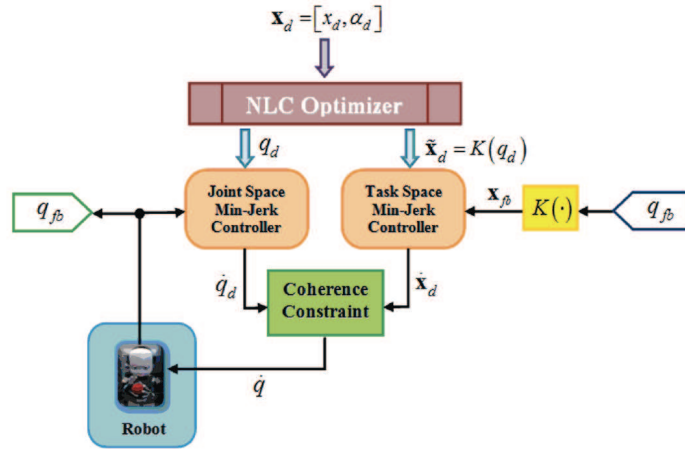


Fig. 8 The multi-referential scheme for trajectory generation. K is the forward kinematics map; q_{fb} is the vector of encoder signals.

in open loop. A complete discussion of the rationale of the modifications to the trajectory generation is outside the scope of this paper; the interested reader is referred to Pattacini et al. [16]. Reasons to prefer a feedback formulation include the possibility of smoothly connecting multiple pieces of trajectories and correcting on line for accumulation of errors due to the enforcement of the constraints of the multi-referential method.

4.1 Validation and further improvements

As earlier for the dynamics, we compared our method with other methods from the literature. The comparison with the method of Hersch et al. [14] was almost immediate since the work was developed on the iCub. This provides the multi-referential approach together with the VITE trajectory generation at no cost. Additionally, we included in the assessment another controller representing a more conventional strategy that uses the Damped Least-Squares (DLS) rule [17] coupled with a secondary task that comprises the joints angles limits by means of the gradient projection method [18]. This solution employs the third-party package Orocos [19], a tool for robot control that implements the DLS approach and whose public availability and compliance with real-time constraints justified its adoption as one of the reference controllers.

In the first experiment we put to test the three selected schemes in a point-to-point motion task wherein the iCub arm was actuated in the “7-DoF mode” and where the end-effector was controlled both in position and orientation. Results show that

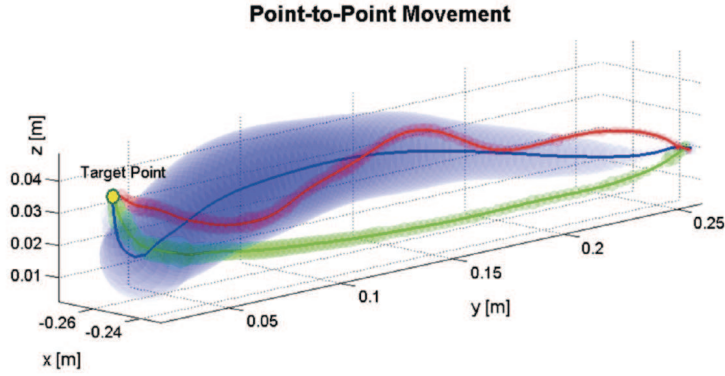


Fig. 9 Point-to-point Cartesian trajectories executed by the three controllers: the VITE-based method produces on average the blue line, the minimum-jerk controller result is in green, the DLS system using Orocos in red. Bands containing all the measured paths within a confidential interval of 95% are drawn in corresponding colors. Controllers settings are: $T = 2.0s$ for the minimum-jerk system, $\alpha = 0.008$, $\beta = 0.002$, $K_P = 3$ for the VITE (see [14] for the meaning of the parameters), and $\mu = 10^{-5}$ for the damping factor of the DLS algorithm.

paths produced by our controller and by the DLS-based system are well restricted in narrow tubes of confidence intervals and are quite repeatable; conversely the VITE is affected by a much higher variability. Figure 9 highlights results for a set of 10 trials of a typical reaching task where the right hand is moved from a rest position to a location in front of the iCub with the palm directed downward.

Table 2 summarizes the measured in-target errors for the three cases: all the controllers behave satisfactory, but the DLS achieves lower errors because operates continuously on the current distance from the target x_d , being virtually capable of canceling it at infinite time. On the contrary, strategies based on the interaction with an external solver bind the controller module to close the loop on an approximation \tilde{x}_d of the real target that is determined by the optimization tolerances as in 11.

Table 2 Mean errors along with the confidence levels at 95% computed when the target is attained. An average measure of the variability of executed path is also given for the three controllers.

Controller	Position error	Orientation error	Mean radius of the trajectory band
VITE	$1.3 \pm 1.4 \cdot 10^{-3} mm$	$0.041 \pm 0.05 rad$	$10 \pm 10.8 mm$
Min-jerk	$3.0 \pm 1.3 \cdot 10^{-3} mm$	$0.048 \pm 0.008 rad$	$2.5 \pm 1.5 mm$
DLS	$1.3 \pm 1.4 \cdot 10^{-3} mm$	$0.016 \pm 0.028 rad$	$2.0 \pm 1.36 mm$

Additional experiments tend to favor our method. For example, measuring the jerk of the resulting trajectory shows a gain of our method by 43% from the VITE

and of about 69% from DLS. This turns out to be crucial for more complicated trajectories when speed factors make the minimum jerk controller even more advantageous.

Further improvements can be made on the quality of the inverse kinematic results by means of machine learning. As for the dynamics, we initially estimated the function K from the CAD models of the iCub. This is a good initial guess in need of refinement. The goal here is therefore to design a procedure that allows enforcing eye-hand coordination such that, whenever the robot reliably localizes a target in both cameras, it can also reach it. Here we further simplified the problem (from the visual point of view) and decided to learn only the position of the end-effector (x, y, z) since the orientation of the hand in the image is difficult to detect reliably. For this problem, the input space is defined by the position of the hand (or the target) in the two cameras (u_l, v_l) and (u_r, v_r) with respect to the current head configuration.

To sum up, having defined the input and the output space, the map M that is to be learned is:

$$(x, y, z)_H = M(u_l, v_l, u_r, v_r, T, V_s, V_g), \quad (12)$$

where $(u_l, v_l, u_r, v_r) \in \mathbb{R}^4$ represent the visual input of the position of the hand in the iCub cameras, whereas $(T, V_s, V_g) \in \mathbb{R}^3$ accounts for the proprioceptive part of the input designating the tilt, the pan and the vergence of the eyes; finally, $(x, y, z)_H \in \mathbb{R}^3$ is the Cartesian position of the hand expressed in the head-centered frame.

This map can be learned by a regression method if enough training samples are available and these can be in turn collected if we can measure (u_l, v_l, u_r, v_r) by means of vision (see section 5). Some preliminary results by using a sigmoidal neural network from Matlab (Neural Network Toolbox) trained with backpropagation can be seen in figure 10. The training phase is carried out off-line. The neural network consists of 7 nodes in the linear input layer, 50 nodes for the hidden layer implemented with the ordinary hyperbolic tangent function and 3 nodes in the linear output layer: an overall number of 15000 samples has been employed for training and validation, whereas 5000 samples have been used for testing. The neural network provides a very good estimation of M as demonstrated by the testing phase. Notably, as expected, the z component estimation is the most affected by noise since it accounts principally for the distance of the hand from the head, a value that is not directly measured by the cameras but only indirectly from binocular disparity. The inspection of the mean and standard deviation supports this claim, i.e. mean error 0.00031m and standard deviation of 0.0055m for the x and y components and about twice as big for z .

As concluding remark, it is relevant to outline here that an upcoming activity has been planned with the purpose to replace the off-line training phase with a fully online version that resorts to random features as in Gijsberts et al. [10] and will eventually make the robot learn the eye-hand coordination completely autonomously.

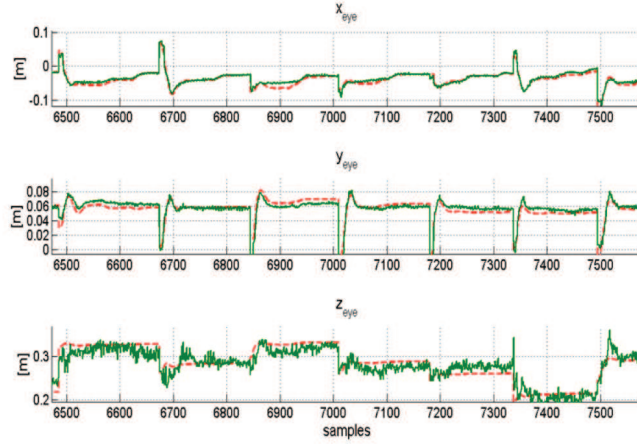


Fig. 10 The desired target (dashed red) and the corresponding outputs of the neural network (green) for the three Cartesian coordinates in the head centered frame.

5 Vision

The remaining piece of information in this journey through the structure of the iCub controller is certainly vision. We strive to provide reliable estimates of object in space since this enables the control of action as presented earlier. One appealing visual cue is motion and we have been recently able to devise a method which provides motion segmentation independent from the movement of the cameras.

Our method is based on the analysis of failures of the standard Lucas-Kanade algorithm [20]. As a general rule, in order to verify that the instant velocity v of a point p has been correctly estimated, the patch W around that point in the image I_t is compared to the patch of the same size at $p + v$ in the new image I_{t+1} (where the original point is supposed to have moved). Given a suitable threshold Θ_M , the discrepancy measure

$$M(p) = \sum_{q \in W} (I_t(p+q) - I_{t+1}(p+v+q))^2, \quad (13)$$

is then used to evaluate whether tracking was correctly performed ($M(p) < \Theta_M$) or not ($M(p) \geq \Theta_M$). It is thus interesting to analyze empirically when the Lucas-Kanade algorithm tends to fail and why. Conclusions from this investigation will lead directly to a method to perform independent motion detection. The main empirical circumstances in which errors in the evaluation process of the optical flow arise are three:

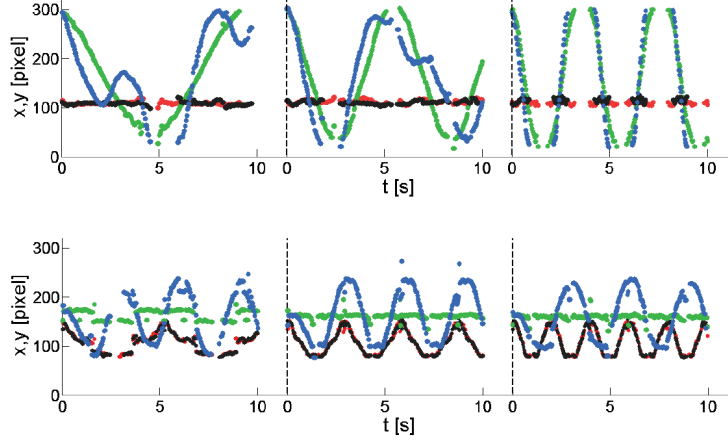


Fig. 11 Trajectories of the x, y coordinates of the center of mass of the areas detected as moving independently. The cart is moving parallel (up) or orthogonal (down) with respect to the image plane. The plots are reported for the following cart speeds: from left to right 20, 40, 100cm/s. Colors legend: (1) green for x and red for y in the case of a static head; (2) blue for x and black for y in the case of a head rotating at $20deg/s$.

- Speed. The instantaneous velocity of the point is too large with respect to the window where motion is being considered. Hence, the computation of temporal derivatives is difficult;
- Rotations. The motion around the point has a strong rotational component and thus, even locally, the assumption regarding the similarity of velocities fails;
- Occlusions. The point is occluded by another entity and obviously it is impossible to track it in the subsequent frame.

Tracking failures caused by high punctual speed depend exclusively on the scale of the neighborhood where optical flow is computed. This issue is usually solved by the so called pyramidal approach which applies the Lucas-Kanade method at multiple image scales. This allows evaluating iteratively larger velocities first and then smaller ones. Instead we determined empirically that when rotations cause failures in the tracking process, this is often a consequence of a movement independent from that of the observer. The third situation in which Lucas-Kanade fails, is caused by occlusions. In this context the main role in determining whether optical flow has been successfully computed is played by the speed at which such occlusion takes place.

We therefore look for points where tracking is likely to fail as soon as one of the conditions discussed is met, i.e. flow inconsistencies due to rotations or occlusions. In detail, we run Lucas-Kanade over a uniform grid on the image, perform the comparison indicated in equation 13 and then filter for false positives (isolated failures). The results is a set of independent moving blobs.

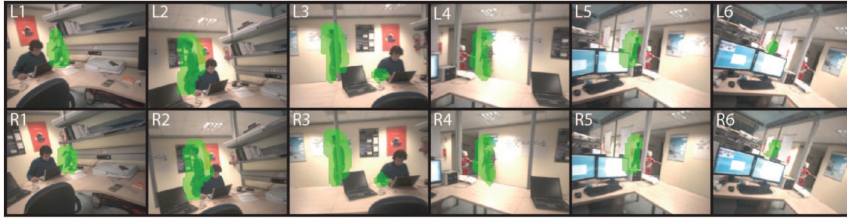


Fig. 12 A sequence of images recorded during the real time stereo tracking of a person walking in front of the iCub: six images are shown in temporal order from L1 to L6 (left camera) and R1 to R6 (right camera). The walking person is highlighted with a green blob using the result of proposed algorithm.

We tested the method both in controlled situations (a small device moving linearly in front of the iCub) and, more generally, in tracking people and other moving objects in the laboratory. Figure 11 shows results of tracking with both stationary and moving cameras (therefore without and with ego-motion respectively). In the configuration considered, a linear speed of 10cm/s corresponds to one pixel per frame in a 30 frames-per-second (fps) acquisition. Experiments were conducted up to 100cm/s and with the iCub head adding movement up to 40deg/s .

The sequence of images in figure 12 is an example of a more naturalistic tracking. In spite of the complexity of the background, it is evident from the images that our method produces robust detection of the moving target with a behavior that varies smoothly in time and is consistent with respect to the two different views acquired from the left and right cameras of the robot. In particular, the movement of the target is effectively tracked both when the person is far from the robot (frames 1 and 6) as well as when he gets closer to it (frames 2-5). Furthermore a substantial modification to light conditions exists with a maximum of brightness reached approximately at frame 4. The algorithm is robust to occlusions: this is visible at the frames in which pillars and posters cover the person. Notably, at frame 3 another person sitting at the table produces a secondary blob with his hand. This distractor is of limited size and it does not interfere with the task since the tracker is instructed to follow the largest blob in the sequence.

These are the data that at the moment the iCub uses for attention, for tracking and which are eventually passed to the reaching controller described earlier. We favored robustness to accuracy here in order to be able to run learning methods and exploration of the environment for considerable periods of time (e.g. as for collecting the 20000 samples mentioned in section 4.1). Our experiments show that this goal has been fully achieved.

6 Conclusions

This paper dealt with the problem of building a reliable architecture to control reaching in a humanoid robot where many degrees of freedom need to be coordinated. We have shown original solutions to vision (using motion), to kinematics (using robust optimization and a multi-referential trajectory formulation) and dynamics (by enabling impedance control from a set of FTSs). Although certain aspects of these methods are somewhat traditional, their specific application and combination is novel. We took particular care in testing all methods rigorously and comparing them with other methods in the literature.

Furthermore, the entire implementation of this software is available, following the iCub policies, as open source (GPL) from the iCub repository. These libraries and modules, besides running on the iCub, are available to the research community at large. The algorithms are almost always embedded in static libraries ready to be picked up by others.

The iCub repository can be found at <http://www.icub.org> and browsed on SourceForge (<http://www.sourceforge.net>). Several videos of the iCub showing the methods described in this paper are available on the Internet and in particular at this site: http://www.youtube.com/watch?feature=player_profilepage&v=LMGSok5tN4A.

Acknowledgements The research described in this paper is the result of work from several EU projects: CHRIS grant agreement number FP7-ICT-215805, Poeticon grant agreement number FP7-ICT-215843, and EFAA grant agreement number FP7-ICT-270490. Many students and post-docs were involved in this research and are gratefully acknowledged: in particular Ugo Pattacini, Carlo Ciliberto, Vadim Tikhonoff, Matteo Fumagalli, Serena Ivaldi, Arjan Gijsberts, and Marco Randazzo.

References

1. Metta G, Natale L, Nori F, et al (2010) The iCub Humanoid Robot: An Open-Systems Platform for Research in Cognitive Development. *Neural Networks*, special issue on Social Cognition: From Babies to Robots. 23:1125–1134
2. Villani L, De Schutter J (2008) Force control – chapter 7. In: Siciliano B, Khatib O (ed) *Springer Handbook of Robotics*, Springer
3. Sciavicco L, Siciliano B (2005) Modelling and Control of Robot Manipulators. *Adv. textbooks in Control and Signal Processing*. Springer, 2nd edition
4. Metta G (2010) The iCub website. <http://www.iCub.org>
5. Metta G, Vernon D, Sandini G (2005) The RobotCub Approach to the Development of Cognition: Implications of Emergent Systems for a Common Research Agenda in Epigenetic Robotics. In: *5th Epigenetic Robotics workshop*, Nara, Japan, July 2005
6. Tilley A R (2002) *The measure of man & woman: human factors in design*. Wiley Interscience
7. Tsagarakis N G, Metta G, Sandini G et al (2007) iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics* 21(10):1151–1175
8. Fitzpatrick P, Metta G, Natale L (2008) Towards Long-Lived Robot Genes. *Robotics and Autonomous Systems* 56(1):29–45

9. Featherstone R, Orin D E (2008) Dynamics – chapter 2. In: Siciliano B, Khatib O (ed) Springer Handbook of Robotics, Springer
10. Gijssberts A, Metta G (2011) Incremental Learning of Robot Dynamics using Random Features. In: IEEE International Conference on Robotics and Automation. Shanghai, China, May 9–13
11. Nguyen-Tuong D, Peters J, Seeger M (2008) Computed torque control with nonparametric regression models. In: American Control Conference (ACC 2008), June 2008, pp. 212-217
12. Rahimi A, Recht B (2008) Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems 20, pp. 1177-1184
13. Wächter A, Biegler L T (2006) On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming* 106(1):25-57
14. Hersch M, Billard A G (2008) Reaching with multi-referential dynamical systems. *Autonomous Robots* 25(1-2):71-83
15. Abend W, Bizzi E, Morasso P (1982) Human arm trajectory formation. *Brain* 105:331-348
16. Pattacini U, Nori F, Natale L, Metta G, Sandini G (2010) An Experimental Evaluation of a Novel Minimum-Jerk Cartesian Controller for Humanoid Robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.1668-1674, Taipei, Taiwan, October 18-22
17. Deo A S, Walker I D (1992) Robot Subtask Performance with Singularity Robustness using Optimal Damped Least-Squares. In: IEEE International Conference on Robotics and Automation
18. Lee H Y, Yi B J, Choi Y (2007) A Realistic Joint Limit Algorithm for Kinematically Redundant Manipulators. In: IEEE International Conference on Control, Automation and Systems
19. The Orocos website. <http://www.orocos.org/kdl>
20. Lucas B.D., Kanade T (1981) An Iterative Image Registration Technique with an Application to Stereo Vision. In: proceedings of IJCAI, pp.674–679