Learning Robust Task Priorities and Gains for Control of Redundant Robots
Luigi Penco, Enrico Hoffman, Valerio Modugno, Waldez Gomes, Jean-Baptiste Mouret, Serena Ivaldi

To cite this version:
Luigi Penco, Enrico Hoffman, Valerio Modugno, Waldez Gomes, Jean-Baptiste Mouret, et al.. Learning Robust Task Priorities and Gains for Control of Redundant Robots. IEEE Robotics and Automation Letters, IEEE 2020, 5 (2), pp.2626-2633. 10.1109/LRA.2020.2972847. hal-02456663

HAL Id: hal-02456663
https://hal.inria.fr/hal-02456663
Submitted on 27 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Learning Robust Task Priorities and Gains for Control of Redundant Robots

Luigi Penco\textsuperscript{1}, Enrico Mingo Hoffman\textsuperscript{3}, Valerio Modugno\textsuperscript{2}, Waldez Gomes\textsuperscript{1}, Jean-Baptiste Mouret\textsuperscript{1}, Serena Ivaldi\textsuperscript{1}

\textbf{Abstract}—Generating complex movements in redundant robots like humanoids is usually done by means of multi-task controllers based on quadratic programming, where a multitude of tasks is organized according to strict or soft priorities. Time-consuming tuning and expertise are required to choose suitable task priorities, and to optimize their gains. Here, we automatically learn the controller configuration (soft and strict task priorities and Convergence Gains), looking for solutions that track a variety of desired task trajectories efficiently while preserving the robot's balance. We use multi-objective optimization to compare and choose among Pareto-optimal solutions that represent a trade-off of performance and robustness and can be transferred onto the real robot. We experimentally validate our method by learning a control configuration for the iCub humanoid, to perform different whole-body tasks, such as picking up objects, reaching and opening doors.

\textbf{I. INTRODUCTION}

Redundant robots like humanoids or mobile manipulators can simultaneously perform several tasks, such as reaching for a target while balancing. The usual way to control these robots is to formulate the problem as a multi-task Quadratic Program (QP) [1], where the tasks are organized according to defined priorities and the goal is to minimize the tasks errors while satisfying a set of constraints, such as joint limits, collisions, etc. Finding the strict task priorities [2], [3], [4], i.e. \textit{null-space relations}, and/or the soft task priorities [5], [6], i.e. \textit{weighted combinations}, together with the task gains is often a time-consuming trial-and-error tuning procedure that requires human expertise and many tests on the robot. Here, we aim at the automated off-line optimization of such parameters for generic multi-task controllers, leveraging multi-objective optimization.

\textbf{A. Related work}

In the literature, many approaches have been proposed to learn soft priorities. In [7], the authors parameterized the weight of each task by radial basis functions defined over time, allowing a temporal adaptation of the different priorities to the specific motions. A derivative-free stochastic optimization is used to learn the optimal parameters. Lober \textit{et al.} [8] propose a framework that modifies a set of initially interfering tasks, via stochastic optimization of their parameters. They use Gaussian kernels to compute variance-dependent weights that allow to handle conflicts between tasks. Paraschos \textit{et al.} [9] use probabilistic movement primitives to learn the accuracy of different tasks. They exploit the variability of demonstrations given in Cartesian and joint spaces as a prioritization criteria to organize the different tasks. In [10], Dehio \textit{et al.} use a mixture of controllers for whole-body motion generation and consider the mixture coefficients as policy parameters, optimizing them by means of a derivative-free stochastic optimization algorithm. Their approach allows the transfer of the learned policy to new tasks, but the optimized mixture coefficients are not verified on the real robot.

For strict hierarchies, solutions have been proposed to overcome discontinuities in the control problem [11], [12], arising when priorities are continuously rearranged. Dehio and Steil [13] proposed a dynamically-consistent generalized hierarchical control that allows choosing, for each pair of tasks, between a soft or a hard priority with one task having null effect on the other one. However, their method is only validated in simulation on a manipulator.

Another combination of soft weighting and hierarchies...
is proposed in [14], where Silvério et al. introduce an identification of demonstrated priority behaviors, given an initial set of candidate task hierarchies, that allows the robot to reproduce the learned priorities in new situations. The demonstrations are given by implementing an inverse kinematics controller with the desired hierarchies in the simulated robot and making it track moving references. Both [13] and [14] limit the number of tasks to three.

Overall, the previous works propose interesting solutions for learning simultaneously soft and hard priorities, but for few tasks and in simulation. This is not enough for the whole-body control of real humanoid robots, where we usually have a higher number of tasks to fulfill. On top of that, learning solutions in simulations does not guarantee that such solutions will work on the real robot: this is a known problem caused by the reality gap.

Achieving transferable solutions is the central focus of many recent works in robotics, [15], [16], [17]. For example, optimal control and movement primitives are combined in [17] to find solutions which can be easily deployed on the real robot. However, they strongly rely on the accuracy of the simulated model to ensure the transferability of solutions. Another common approach that can help achieving the adaptability of the learned solution on new scenarios is Domain Randomization (DR) [16], which consists in randomizing some aspects of the simulation to enrich the range of possible environments experienced by the learner. In [15], robust policies for pivoting a tool held in the robot's gripper were learned in simulation, given random friction and control delays, such that the learned policies proved to be effective on the real robot as well. In [18], the learning procedure guarantees strict constraints fulfillment, but transferring knowledge from simulation to reality did not fully achieve the desired behavior. This implies that constraints satisfaction can be beneficial, but not enough to achieve transferability. Indeed to achieve a better generalization, solutions which are robust rather than optimal are needed. For instance, Del Prete et al. [19] proposed to improve the robustness of task space inverse dynamics by modeling uncertainties in the joint torques of humanoid robots. Charbonneau et al. instead, learned the soft priorities that optimize for both task performance and robustness [20], by applying the idea of DR. In [15], robust policies for pivoting a tool held in the robot's gripper were learned in simulation, given random friction and control delays, such that the learned policies proved to be effective on the real robot as well. In [18], the learning procedure guarantees strict constraints fulfillment, but transferring knowledge from simulation to reality did not fully achieve the desired behavior. This implies that constraints satisfaction can be beneficial, but not enough to achieve transferability. Indeed to achieve a better generalization, solutions which are robust rather than optimal are needed. For instance, Del Prete et al. [19] proposed to improve the robustness of task space inverse dynamics by modeling uncertainties in the joint torques of humanoid robots. Charbonneau et al. instead, learned the soft priorities that optimize for both task performance and robustness [20], by applying the idea of DR.

To summarize, we address the problem of automated learning of controllers for redundant robots with the following contributions: first, we introduce a parametrization of strict task priorities, so to learn simultaneously the hierarchical structure and the parameters of the controller with a multitude of concurrent tasks; second, we use multi-objective optimization algorithm to get a set of Pareto-optimal solutions (controller configurations) to be tested by the user on the real robot.

We validate our optimization process on the humanoid robot iCub, for double support motions. We compare controllers that were automatically optimized on a training sequence of motions against a baseline hand-tuned controller, over different motion sequences (see Figure 1). We further show that our optimized controller can be successfully employed to execute a variety of motions, other than those used in the training, for example for teleoperation, where reference trajectories (unknown a priori) are generated in real-time by the human operator. Section V details our experiments, which are also shown in the attached video.

II. BACKGROUND: Multi-task QP control

For a given task $T_k$, the QP control consists in solving the following optimization problem at each control time step:

$$
\begin{align*}
\min_{u} & \quad \|A_k u - b_k\|^2 + \epsilon \|u\|^2 \\
\text{s.t.} & \quad c \leq C \ u \leq \tau \\
& \quad u_{\min} \leq u \leq u_{\max}
\end{align*}
$$

(1)

where $u$ is the control input vector, e.g. the joint velocities $\dot{q}$ for Inverse Kinematics (IK) control, or the joint torques $\tau$ for Inverse Dynamics (ID) control; $A_k$ is the equivalent Jacobian matrix of the task $k$, e.g. the Jacobian matrix $J_k$ for IK control or $J_k B^{-1}$ for ID control (with $B$ being the inertia matrix); $b_k$ the reference value for the task, $\epsilon$ a regularization factor used to handle singularities, $u_{\min}$ and $u_{\max}$ the control input limits and $c \leq C \ u \leq \tau$ are other equality and inequality constraints, e.g. dynamics, collision avoidance, and contact related constraints.
If we consider $n$ levels of hierarchies, we can solve $n$ QP problems including local equality constraints to ensure the strict priorities between the tasks in each hierarchy, i.e. $A_{i-1}u_{i-1} = A_{i-1}u$, ..., $A_0u_0 = A_0u$. At a given level $i$ of the hierarchy, we can also consider a weighted combination of tasks:

$$\min_u \sum_k w_k (\|A_k u - b_k\|^2 + \epsilon \|u\|^2)$$

$$s.t. \quad \varepsilon \leq Cu \leq \varepsilon$$

$$u_{min} \leq u \leq u_{max}$$

(2)

For each task $T_k$, a reference is defined, e.g. in IK control $b_k = \hat{p}_k + \lambda_k e$, where $\hat{p}_k$ is a feed-forward velocity term and $e$ is a Cartesian pose error, computed as in [23], multiplied by the Convergence Gain $\lambda_k$; while in Inverse Dynamics control $b_k = \ddot{p}_k - \dddot{J}_k \dot{q} + \lambda_k^P e + \lambda_k^D \dot{e}$, where $\dddot{p}_k$ is the desired task acceleration, $\dddot{J}_k$ the Cartesian velocity error, $\dot{J}_k$ the derivative of the Jacobian of task $k$, $\dot{q}$ the derivative of the actual joint configuration and $\lambda_k^P, \lambda_k^D$ the Convergence Gains.

We define as stack $S$ a set of $n$ tasks $\{T_1, \ldots, T_n\}$ organized according to different soft or strict priorities:

$$S = (w_1 T_1 + \ldots + w_i T_i) / \ldots / (w_j T_j + \ldots + w_n T_n);$$

(3)

where $T_a + T_b$ means that the tasks $T_a$ and $T_b$ are in a soft priority relation with $w_a, w_b$ being the corresponding weights, while $T_a/T_b$ means that the two tasks are in a strict priority relation ($T_b$ acts in the null space projection of $T_a$).

**Definition:** We define as “control configuration” of a controller with $n$ tasks $\{T_1, \ldots, T_n\}$, the set of the Soft Priorities Weights $w_k$, the task Converge Gain $\lambda_k$ and the strict priority relations between the tasks, formalized as the level of each task $T_k$ in their stack $S$.

### III. Method

The goal of our method is to automatically learn a control configuration of a controller enabling the real robot to execute a variety of different reference motions (for a given category of movements, e.g. double support). We look for a “generic” control configuration that trades-off performance on several motions, rather than a “motion-specific” controller that is highly optimized for a given movement but requires the optimization to be re-run for every other motion. We also want the result of the optimization procedure to be transferable onto the real robot.

Figure 2 is a flowchart of the learning procedure. First, a multi-objective optimization process based on roll-outs performed on a simulated robot model computes off-line a set of Pareto-optimal solutions. Each solution is a control configuration that trades-off different performance criteria (i.e., tracking score and a robustness/balancing score), optimized on a training sequence. Then the user selects candidate solutions from the Pareto front and validates them on the real robot using test motion sequences. Here, the purpose is to find transferable solutions that are high-performing and well-
TABLE I: Considered tasks, associated symbols and control parameters (Soft Priority Weight (SPW), Hierarchy Level Selector (HLS) and Converge Gain (CG)) for the controllers C1 and C2

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Symbol</th>
<th>SPW</th>
<th>HLS</th>
<th>CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>left foot pose</td>
<td>T_{lf}</td>
<td>w_f</td>
<td>l_f</td>
<td>\lambda_{f,foot} \sigma_{foot}</td>
</tr>
<tr>
<td>right foot pose</td>
<td>T_{rf}</td>
<td>w_f</td>
<td>l_f</td>
<td>\lambda_{f,foot} \sigma_{foot}</td>
</tr>
<tr>
<td>left hand position</td>
<td>T_h</td>
<td>w_h</td>
<td>l_h</td>
<td>\lambda_{hand}</td>
</tr>
<tr>
<td>right hand position</td>
<td>T_h</td>
<td>w_h</td>
<td>l_h</td>
<td>\lambda_{hand}</td>
</tr>
<tr>
<td>com height</td>
<td>T_c</td>
<td>w_c</td>
<td>l_c</td>
<td>\lambda_{com}</td>
</tr>
<tr>
<td>com (x,y)</td>
<td>T_{cxy}</td>
<td>w_{cxy}</td>
<td>l_{cxy}</td>
<td>\lambda_{com}</td>
</tr>
<tr>
<td>waist orientation</td>
<td>T_{wso}</td>
<td>w_{wso}</td>
<td>l_{wso}</td>
<td>\sigma_{waist}</td>
</tr>
<tr>
<td>waist height</td>
<td>T_{wh}</td>
<td>w_{wh}</td>
<td>l_{wh}</td>
<td>\lambda_{waist}</td>
</tr>
<tr>
<td>head orientation</td>
<td>T_h</td>
<td>w_h</td>
<td>l_h</td>
<td>\sigma_{head}</td>
</tr>
<tr>
<td>chest orientation</td>
<td>T_c</td>
<td>w_c</td>
<td>l_c</td>
<td>\sigma_{chest}</td>
</tr>
<tr>
<td>neck posture</td>
<td>T_n</td>
<td>w_n</td>
<td>l_n</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>torso posture</td>
<td>T_t</td>
<td>w_t</td>
<td>l_t</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>left arm posture</td>
<td>T_a</td>
<td>w_a</td>
<td>l_a</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>right arm posture</td>
<td>T_a</td>
<td>w_a</td>
<td>l_a</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>left lower arm posture</td>
<td>T_{la}</td>
<td>w_{la}</td>
<td>l_{la}</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>right lower arm posture</td>
<td>T_{la}</td>
<td>w_{la}</td>
<td>l_{la}</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>left leg posture</td>
<td>T_l</td>
<td>w_l</td>
<td>l_l</td>
<td>\mu_{posture}</td>
</tr>
<tr>
<td>right leg posture</td>
<td>T_r</td>
<td>w_l</td>
<td>l_l</td>
<td>\mu_{posture}</td>
</tr>
</tbody>
</table>

 balanced / “robust”, i.e., reducing the tipping moment of the robot. In the following, we detail all the steps of the proposed method.

A. Motion Generation and Task Specification

The Motion Generator is a module generating a sequence of robot movements, defined by reference trajectories \( b_k \). Any motion planning algorithm could be used. In this paper, without losing generality, we use our motion retargeting framework [24], which allows us to generate references for the End Effectors (EEs) both in the Cartesian and postural space from the imitation of human movements.

The tasks \( T_k \) considered for the controller are those for which the motion generator produces the references \( b_k \).

B. Control Configuration – parameters to optimize

For each task \( T_k \), we want to learn its Soft Priority Weight (SPW) \( w_k \), its Convergence Gains (CG) (position error gain \( \lambda_k \) and orientation error gain \( \sigma_k \) for Cartesian tasks, postural error gain \( \mu_k \) for joint tasks) and its position in the stack/hierarchy. A specific task \( T_k \) can be either active in one level \( S \) of the stack \( S \) or completely deactivated (i.e., not in the stack). The parameter \( l_k \) called Hierarchy Level Selector (HLS) encodes the activation of the \( k \)-th task in \( S \) as follows:

\[
\begin{align*}
T_k & \subseteq S_1 \quad \text{if} \quad 0 \leq l_k \leq 0.25 \\
T_k & \subseteq S_2 \quad \text{if} \quad 0.25 < l_k \leq 0.5 \\
T_k & \subseteq S_3 \quad \text{if} \quad 0.5 < l_k \leq 0.75 \\
T_k & \text{ deactivated} \quad \text{if} \quad 0.75 < l_k \leq 1
\end{align*}
\]

Since humanoid robots exhibit bilateral symmetry, we used the same values for the weights \( w_k \) and the activation levels \( l_k \) for tasks related to the left and right parts of the robot. The control parameters all consist of real values \( \in [0,1] \).

C. Learning Algorithm (offline optimization on simulated robot)

The main features that should characterize our “generic” controller are high-performance (tracking of the reference motions) and robustness (reduction of the tipping moment of the robot during the motion). These two objectives may be antagonistic (e.g., when the desired motions are particularly challenging for the robot balance) hence the problem is naturally posed as a multi-objective optimization problem.

Multi-objective optimization relies on the Pareto dominance concept [25]: a solution \( x_1 \) dominates \( x_2 \) if and only if \( x_1 \) is better than \( x_2 \) for all the objectives; if \( x_2 \) is better for at least one objective, then \( x_1 \) and \( x_2 \) are equally interesting as they represent different trade-offs. Using this definition, optimizing means finding the set of the non-dominated solutions of the search space, that is, solutions that cannot be improved with respect to one objective without decreasing their score with respect to the other. This set is called the “Pareto front”. Although we select a single solution to be used on the robot, the knowledge of multiple Pareto-optimal solutions, “helps the user to compare, choose a trade-off solution, avoiding multiple optimization runs and artificial fix-ups” [25].

The algorithm we opted for is the Non-dominated Sorting Genetic Algorithm II (NSGA II) [26], one of the most efficient stochastic multi-objective optimization methods. The objective functions are the accumulated tracking error \( f_1 \), and a robustness score \( f_2 \), i.e. a measure of the ZMP position inside the Support Polygon (SP). We additionally considered a third objective function \( f_3 \), consisting of a fall avoidance score. \( f_3 \) is not strictly necessary since this information is already encoded by \( f_1 \) and \( f_2 \), but we included it to avoid getting initially stuck in some local optimum, as we explain.

\[^2\text{In this work we consider only three levels of priorities, which is a reasonable assumption in humanoids' controllers, especially when dealing with a large number of tasks (TABLE I). However, one could optimize for as many levels as they wish.}\]
later on. The goal of the algorithm is to minimize \( f_1 \) and \( f_2 \) while maximizing \( f_3 \)

\[
\text{Minimize } (f_1(x), f_2(x), -f_3(x)),
\]

where \( x \) are the control parameters. These objectives are never combined together, since we use Pareto-based multi-objective optimization. The objective functions are initialized at zero and change at each time step \( i \) as follows:

\[
\begin{align*}
    f_1 &= f_{1,i-1} + \sum_k \Phi^k_i - \Phi^k_i \\
    f_2 &= f_{2,i-1} + x_{SP} + y_{SP} & \text{if (robot fallen)} \\
    f_2 &= f_{2,i-1} + |x_{\text{f}i}^z| + |y_{\text{f}i}^z| & \text{if (robot not fallen)} \\
    f_3 &= f_{3,i-1} - \alpha_3 & \text{if (robot fallen)}
\end{align*}
\]

(5)

where \( \Phi^k_i \) and \( \Phi^k_i \) are respectively the reference and the actual value measured on the simulated robot of task \( k \) at time step \( i \); \( x_{\text{f}i}^z \) is the distance in the frontal direction of the ZMP from the center of the SP at time step \( i \), while \( y_{\text{f}i}^z \) is the distance in the horizontal direction of the ZMP from the line connecting the feet; \( x_{SP}, y_{SP} \) are the dimensions of the SP and \( \alpha_3 \) an arbitrary positive value. The cost function \( f_1 \) includes both position, orientation and postural tracking errors, which have to be properly combined. The Cartesian positions are computed in \( \text{cm} \); for postural and orientation tasks, we consider for each angle the arc length in \( \text{cm} \) of the circle having as radius the child link of the joint (for postural tasks) or the distance from the global frame to the link frame (for Cartesian orientation tasks). In this way, each task error in \( f_1 \) has approximately the same relative impact in the Cartesian space.

When the algorithm generates some parameters that correspond to an unfeasible stack (e.g. empty level in the hierarchy that is not the last) the controller fails, i.e. it cannot be initialized. It can also fail when the solver cannot find a solution for the \( QP \) problem without violating its constraints. In such cases, we add the following penalties:

\[
\begin{align*}
    f_1 &= f_{1,i-1} + \alpha F_1 (I - i) & \text{if } (i \neq 0) \\
    f_2 &= f_{2,i-1} + \alpha F_2 (I - i) & \text{if } (i \neq 0) \\
    f_3 &= f_{3,i-1} - \alpha F_3 (I - i) & \text{if } (i \neq 0) \\
    f_1 &= F_{\text{max}} & \text{if } (i = 0) \\
    f_2 &= F_{\text{max}} & \text{if } (i = 0) \\
    f_3 &= -F_{\text{max}} & \text{if } (i = 0)
\end{align*}
\]

(6)

where \( I \) is the total number of steps in the recorded sequence, \( i \) is the time step when the solver fails, \( F_{\text{max}} \) is the largest value a floating-point variable can hold, \( \alpha F_3 \) can be set to any arbitrary value greater than \( \alpha_3 \) and lower than \( F_{\text{max}} / I \) and \( F_{\text{max}} / I \), \( F_{\text{max}} / I \), and \( F_{\text{max}} / I \) to arbitrary values that are lower than \( F_{\text{max}} / I \) and respectively greater than the largest increment \((\Delta_1, \Delta_2)\) that \( f_1, f_2 \) could get in a time step when the controller does not fail. Since it might be difficult for the user to determine in advance the lower bound values of \( \alpha F_1, \alpha F_2 \), we consider also \( f_3 \) in the optimization to help the algorithm even for a wrong guess by the user of this lower bound of \( \alpha F_1, \alpha F_2 \). If the first generations indeed have large \( f_1, f_2 \) scores with \( \alpha F_1, \alpha F_2 \) lower than \( \Delta_1, \Delta_2 \), it may happen that the algorithm prefers failing solutions to bad performing solutions that do not fail, hence the controller fails very often and the algorithm struggles to converge to a stable configuration. In such cases, even a solution that ends up in falling represents a temporary dominant solution and can help the algorithm progress toward better solutions. Also, \( f_3 \) drives the search toward “not falling” solutions. With the adopted cost functions design, the penalty factors \( \alpha F_1, \alpha F_2, \alpha F_3, \alpha_3 \) can be set to arbitrary values comprised in between lower and upper bounds that are known, with the exception of the lower bounds of \( \alpha F_1, \alpha F_2 \) which can be easily guessed from the problem setting, since the first is related to the maximum tracking error and the second to the maximum robustness score in a single time step.

D. Solution Selection (online tests, on the real robot)

From (5), (6) we can never get simultaneously a bad score for \( f_3 \) and a good score on \( f_1 \) and \( f_2 \), hence all the final dominant solutions composing the Pareto front have \( f_3 = 0 \), i.e. the robot does not fall. We can then analyze the 2D Pareto front related to \( f_1 \) and \( f_2 \) (Figure 3). We start from the most high-performing solution (i.e. lowest \( f_1 \) and highest \( f_2 \)) and try if the solution works on the real robot. If not, we move progressively to solutions that are less high-performing and more “robust” until we get to a solution that achieves balance on the real robot.

IV. EXPERIMENTS

Robot control: Experiments were performed with the iCub robot, using 32 DoFs for whole-body control. The whole-body controller is developed using the control software library OpenSoT [1], [27]. We tested our approach on an IK-based implementation of the multi-task \( QP \) controller (Section III). For the experiments on the real robot, the constraints considered in the \( QP \) are joint position and velocity limits, and the center of mass (CoM) kept inside the support polygon. To learn the control parameters, the robot is simulated using the open-source simulator Dart [28].

Motion generator, training and testing sequences: To generate the training and testing motion sequences for the optimization and validation of the controller, we used our motion retargeting framework [24]. We recorded the movements of a human equipped with the XSens MVN motion tracking suit, and then we retargeted the motions onto the robot model. For the training set, we recorded a 68-seconds sequence, consisting of a different series of whole-body movements that solicit as many body parts as possible, so to generalize well to other motions. For the validation, we recorded three different sequences: \( S1 \): a squat motion; \( S2 \): a movement where the robot has to shift completely its weight on the left foot and reach a high position with the left hand; \( S3 \): a complex movement where the robot has to

\[3\]If the user chooses values that are too small, the inclusion of \( f_3 \) allows the algorithm to converge to a solution anyways, as explained before. At any rate these parameters do not need an iterative tuning.

...
controllers $C_1$ and $C_2$ to be optimized:} To show that our approach does not depend on the number or type of tasks (e.g., Cartesian, postural), we seek solutions for two different types of controllers, namely $C_1$ and $C_2$. They are specified by the set of tasks considered for their control configuration, reported in TABLE IV. $C_1$ takes as references (from the motion generator) mainly Cartesian tasks, while $C_2$ mostly postural trajectories. Both controllers can be used by the humanoid to realize double support motions, but $C_2$ replicates the controller that we have been using for teleoperating the iCub robot to perform human-like motions in our previous work [24], where the postural tasks are critical for imitating the human. For this reason, we will validate both controllers on the real robot with the aforementioned sequences $S_1$, $S_2$, and $S_3$, but only $C_2$ will be further validated for the robot teleoperation.

**Optimization on training sequences (offline):** The parameters of the two controllers $C_1$ and $C_2$ are learnt by the algorithm NSGA-II implemented in Sferesv2 [29], a C++ framework for multi-core optimization. We set a population $p$ of 100 individuals with 300 generations $g$ (for a total of 30100 evaluations). To provide statistically significant results, we executed in parallel 20 runs on an Intel® Xeon™ E5-2620 with 32 cores at 2.1 GHz. The parallel optimization takes about 15 hours.

The duration of the optimization mainly depends on the length of the learning sequence that has to be simulated for a number of $p \cdot g$ times. We opted for a long learning sequence to find a control configuration that can generalize well to many tasks and that can be found by running the optimization just once. For the penalties in the cost functions we used $\alpha_1 = 0.5$ and $\alpha_3 = 0.1$ (see Section III-C). The algorithm converges to a set of Pareto-optimal trade-offs among $f_1$ and $f_2$ (see Figure II).

**Validation on the test sequences and selection of the solutions (online, on real robot):** Once the Pareto-optimal trade-off solutions are found, we follow the testing procedure of Section III-D to select the final transferable solutions for $C_1$ and $C_2$ on the real robot. We use the three test sequences $S_1$, $S_2$, $S_3$. We tried on the real robot different solutions from the Pareto front starting from those associated to the lowest tracking error III-D). In the attached video, we show on the learned (median) controller $C_1$ with a squat motion with straight torso reference, that these solutions are not robust enough to be transferred onto the real robot, which fails. After trying those with $f_2 = 37$, $f_2 = 32$ and $f_2 = 27$ we found that the median solution with associated robustness score $f_2 = 22$ is transferable to the real robot (see attached video). We report here the structure of the stack of $C_1$ that represents the most frequent solution, given $f_2 = 22$ (see

\[ \mathcal{S}_{C_1} = \frac{(w_f(T_{ff} + T_f) + w_n T_n)}{(w_{cxy} T_{cxy} + w_{cxy} T_{cxy} + w_n T_n + w_{cxy} T_{cxy})} \]

(7)

while for $C_2$ we have:

\[ \mathcal{S}_{C_2} = \frac{(w_f(T_{ff} + T_f) + w_n T_n + w_n T_n)}{(w_{cxy} T_{cxy} + w_{cxy} T_{cxy} + w_f T_f + w_n (T_{ff} + T_{ff}))} \]

(8)

The corresponding median gains and soft weights are reported in TABLE II and III along with the interquartile range (IQR) of these parameters in the 20 learned configurations. TABLE IV indicates the frequency of each task in the different levels of the hierarchy in the 20 runs.

**Comparison with a baseline hand-tuned controller:** We compare the performance of the learned controller for $C_1$ with respect to a baseline solution that was manually tuned by an expert. The test is performed on the three different sequences $S_1$, $S_2$, $S_3$.

The baseline is the following Hand-tuned controller (HT):

\[ \mathcal{S}_{HT} = \frac{(w_{n}(T_{ff} + T_{ff}) + T_{efy} + T_n)}{(T_{cxy} + T_f + (T_{ff} + T_{ff}))} \]

(9)

where the tasks concerning the feet contacts $T_{ff}$, $T_f$

\[^5\text{The purpose of the 20 runs is to provide statistical evidence about the convergence of the multi-objective optimization algorithm. We can notice that the variability of the Pareto front is small (see IQR region in Figure 3), meaning that for a real use-case only one run is sufficient to compute the Pareto front and to find the Pareto-optimal solutions.}\]
and the center of mass $T_{cxy}$ are reasonably considered at the highest level of priority, being the most relevant for balance. The $HT$ solution is validated on the robot for double support motions. Soft priorities are all set to 1. The CG are all set to 0.1 except for $\lambda_{feet}$, $\sigma_{feet}$, $\lambda_{com}$ that are set to 1 as done in previous work where our optimization approach was not yet available [24], [1], [27].

Figure 4 and the video show respectively in simulation and on the real robot, how the median learned controller $C_1$ outperforms $HT$ in tracking the sequence $S1$. Since simultaneous mastering of multiple tasks during motion can be challenging (especially if these references are computed on approximate models that do not match reality), some tasks can be tracked with less precision than others. Indeed, the optimization algorithm found that penalizing the tracking of a not optimal reference for the $x$ position of the CoM, can then lead to a better overall whole-body tracking performance, especially a better tracking of the height of the CoM (see Figure 4). In $S2$, by setting $w_{ht} = 0.7$, we can get with $HT$, a performance that is comparable to the median learned controller $C1$ in simulation (see Figure 5). However, when transferring the result onto the real robot, $HT$ makes the robot fall (due to a not robust enough choice of the priorities and CG) while $C1$ does not, as shown in the attached video. In $S3$, $HT$ fails to find a solution, for both $w_{ht} = 0.7, 1$. We then consider the same structure of the stack and the same soft weights of $C1$ in $HT$ to see how only the CG can affect the tracking performance (Figure 6). Also in this case, with the gains set manually (that are not “robust”) the real robot falls, while with the optimized gains it keeps the balance (attached video).

Validation of $C2$ for teleoperation: To show how a controller optimized through our approach can be used for generic motions – not previously encountered in the learning sequence, though from the same category (i.e., double support) – we used the optimized controller of $C2$ in our teleoperation framework [30]. We teleoperate the iCub to perform several actions: spacing the legs and picking up a box, pushing a ball in a box, pushing aside a box, opening and closing the door of a container, dancing and hitting a ball (see Figure 7). The human operator generates the movements in real-time, his movement being tracked by the Xsens MVN suit and retargeted to the robot. The teleoperated sequence, lasting more than 2 minutes of continuous movements, is reported in the video attachment.
Fig. 7: The iCub robot controlled with the learned configuration $C^2$ while performing different teleoperated tasks.

Generality w.r.t. to other robots: Our approach is not robot-specific. To show that it can be applied to other robots with a different kinematics, we optimize the $C^1$ controller for the COMAN robot. The solution is shown in simulation for a short squat motion, in the video attachment.

V. CONCLUSIONS

We proposed a framework to automatically learn both the structure and the parameters of a “generic” whole-body controller. We used multi-objective optimization to look for solutions on the Pareto front, representing optimal trade-offs of performance (tracking the reference trajectories) and robustness (limiting the tipping moment of the robot). This allowed us to efficiently find a set of Pareto-optimal trade-off solutions in simulation. The user can conveniently choose from the Pareto-front a pre-optimized solution to test on the real robot: this reduces considerably the number of test trials on the real robot and improves the parameters tuning. To give the reader an example, to find a suitable controller configuration for humanoid teleoperation it only took us 3 test trials on the real robot. Before our method, tuning such a controller was a time consuming trial-and-error procedure.

In this work we limited our approach to double support motions with a simple QP controller structure that does not include friction cones, contact wrenches and centroidal dynamics in the constraints [31]. Those will be added in future work, to deal with more complex and dynamics motions. For example, we are automatically optimizing the parameters of a controller suitable for walking, where a model predictive controller is used as motion generator for the gait and footstep trajectories.

REFERENCES