Learning Robust Task Priorities and Gains for Control of Redundant Robots

Luigi Penco, Enrico Mingo Hoffman, Valerio Modugno, Waldez Gomes, Jean-Baptiste Mouret, Serena Ivaldi

Learning Robust Task Priorities and Gains for Control of Redundant Robots

Luigi Penco[1], Enrico Mingo Hoffman[3], Valerio Modugno[2], Waldez Gomes[1], Jean-Baptiste Mouret[1], Serena Ivaldi[1]

***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.**

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. *null-space relations*, and/or the soft task priorities [5], [6], i.e. *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.

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 *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 *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 *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

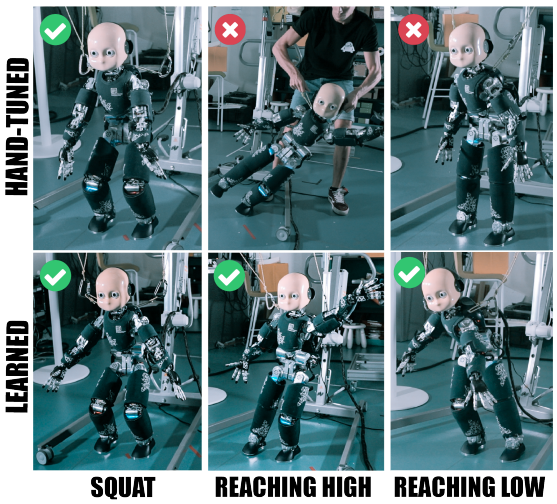

Fig. 1: Despite being tuned to perform several motions in simulation, a hand-tuned controller can easily fail when transferred on the real humanoid and when performing different tasks that challenge its balance. With our multi-objective optimization approach, the learned controller can work on the real robot iCub, also on different test motions.

*This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 731540 (project AnDy) and from the European Research Council (ERC) under Grant Agreement No. 637972 (project ResiBots).

[1] Inria, Loria, Université de Lorraine, CNRS, Nancy, France, serena.ivaldi@inria.fr

[2] Universitá La Sapienza, Roma, Italy

[3] Istituto Italiano di Tecnologia (IIT), Genova, Italy

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. However, both [19] and [20] lack experimental validation on the real robot.

B. Contributions

Differently from previous work, here we propose to learn both the structure of the controller (i.e., the position of each task in the hierarchy) and all its parameters (i.e., the task weights and Convergence Gains). We optimize for a large number of tasks and introduce a parametrization of the strict priorities that allows us to optimize them as simply as the soft weights. We look for solutions that are both high-performing in executing desired trajectories (low tracking error) and "robust" (reducing the tipping moment of the robot).

Any state-of-the-art motion generator [21] still synthesizes dynamically balanced trajectories for inaccurate robot models, without guarantees that a perfect tracking of these desired trajectories is possible on the real robot. At the same time, reactive whole-body controllers have often a very conservative design in terms of balancing, which prevents the robots to track desired trajectories that are instantaneously challenging for balance. It seems appropriate to reason in terms of compromise between tracking performance and balancing/robustness to look for controllers that can effectively work on real humanoid robots.

Instead of arbitrarily combining the two objectives (performance and robustness) into a single cost function, we follow a multi-objective optimization approach: we seek to obtain in a single optimization run the set of the Pareto-optimal solutions, i.e., the optimal trade-offs between the objectives. This enables the robot user to test candidate controllers from the Pareto front directly on the robot, without requiring further optimization or manual tuning. The proposed method is detailed in Section III.

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 IV details our experiments, which are also shown in the attached video.

II. Background: Multi-task QP control

For a given task $\mathcal{T}_k$, the QP control consists in solving the following optimization problem at each control time step:

$$\begin{aligned} \min_{\boldsymbol{u}} \ & \|\boldsymbol{A}_k\boldsymbol{u} - \boldsymbol{b}_k\|^2 + \epsilon\|\boldsymbol{u}\|^2 \\ s.t. \ & \ \underline{\boldsymbol{c}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{c}} \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max} \end{aligned} \tag{1}$$

where $\boldsymbol{u}$ is the control input vector, e.g. the joint velocities $\dot{\boldsymbol{q}}$ for Inverse Kinematics (IK) control, or the joint torques $\boldsymbol{\tau}$ for Inverse Dynamics (ID) control; $\boldsymbol{A}_k$ is the equivalent Jacobian matrix of the task k, e.g. the Jacobian matrix $\boldsymbol{J}_k$ for IK control or $\boldsymbol{J}_k\boldsymbol{B}^{-1}$ for ID control (with $\boldsymbol{B}$ being the inertia matrix); $\boldsymbol{b}_k$ the reference value for the task, ϵ a regularization factor used to handle singularities, $\boldsymbol{u}_{min}$ and $\boldsymbol{u}_{max}$ the control input limits and $\underline{\boldsymbol{c}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{c}}$ are other equality and inequality constraints, e.g. dynamics, collision avoidance, and contact related constraints.

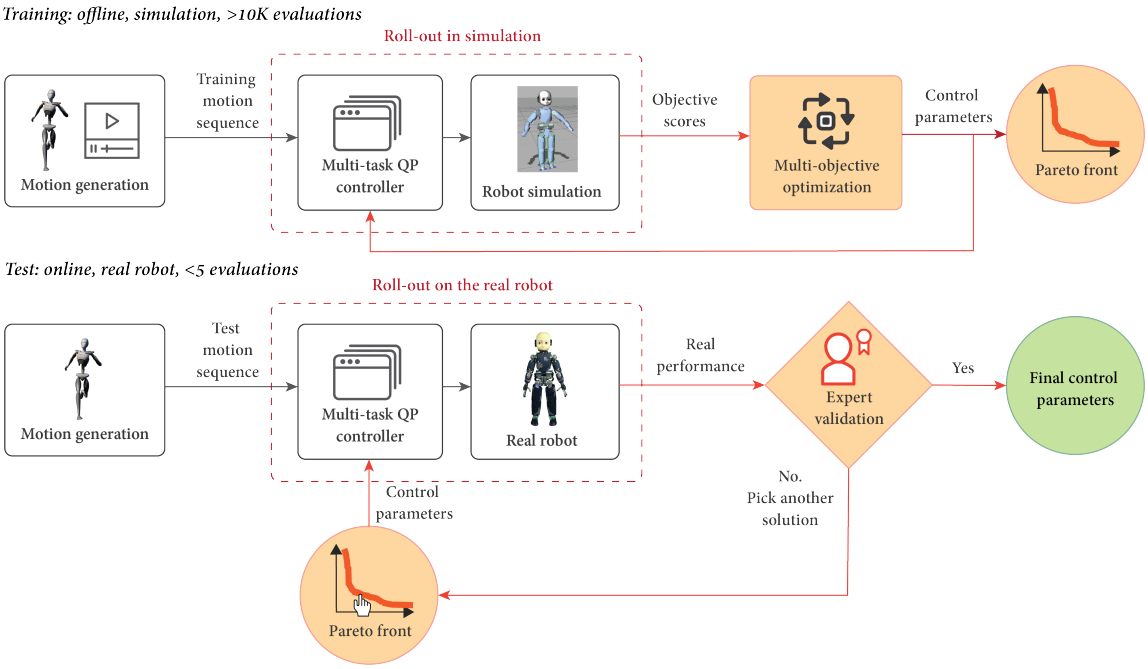

Fig. 2: Overview of the approach. Top: Optimization process (offline). A training motion sequence is generated. The optimization algorithm searches for the best controller configurations that make the simulated robot execute the reference motion sequence. The algorithm computes a set of Pareto-optimal control configurations, i.e. optimal trade-offs between robustness and performance. Bottom: Testing process (online). The user selects the most appropriate control configuration from the Pareto-optimal solutions for the real robot, getting a valid working solution in few trials.

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, i.e. $\boldsymbol{A}_{i-1}\boldsymbol{u}_{i-1} = \boldsymbol{A}_{i-1}\boldsymbol{u}$, ... , $\boldsymbol{A}_0\boldsymbol{u}_0 = \boldsymbol{A}_0\boldsymbol{u}$[1]. At a given level i of the hierarchy, we can also consider a weighted combination of tasks:

$$\begin{aligned} \min_{\boldsymbol{u}} \sum_k & w_k(\|\boldsymbol{A}_k\boldsymbol{u} - \boldsymbol{b}_k\|^2 + \epsilon\|\boldsymbol{u}\|^2) \\ s.t. \quad & \underline{\boldsymbol{c}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{c}} \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max} \end{aligned} \tag{2}$$

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

We define as stack $\mathcal{S}$ a set of n tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ organized according to different soft or strict priorities:

$$\begin{aligned} \mathcal{S} = (w_1\mathcal{T}_1 + ... + w_i\mathcal{T}_i)/ \\ \vdots \\ (w_j\mathcal{T}_j + ... + w_n\mathcal{T}_n); \end{aligned} \tag{3}$$

where $\mathcal{T}_a + \mathcal{T}_b$ means that the tasks $\mathcal{T}_a$ and $\mathcal{T}_b$ are in a soft priority relation with w_a, w_b being the corresponding weights, while $\mathcal{T}_a/\mathcal{T}_b$ means that the two tasks are in a strict priority relation ($\mathcal{T}_b$ acts in the null space projection of $\mathcal{T}_a$).

Definition: We define as **"control configuration"** of a controller with n tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, the set of the Soft Priorities Weights w_k, the task Converge Gain λ_k and the strict priority relations between the tasks, formalized as the level of each task $\mathcal{T}_k$ in their stack $\mathcal{S}$.

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-

[1]Notice that there are also other ways to solve hierarchical problems using a single QP instead of a cascade of QPs [22], however this goes out the scope of the present work.

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

Task		**Control Parameters**		
Description	Symbol	SPW	HLS	CG
left foot pose	$\mathcal{T}_{lf}$	w_f	l_f	$\lambda_{feet}, \sigma_{feet}$
right foot pose	$\mathcal{T}_{rf}$	w_f	l_f	$\lambda_{feet}, \sigma_{feet}$
left hand position	$\mathcal{T}_{lh}$	w_{ha}	l_{ha}	λ_{hand}
right hand position	$\mathcal{T}_{rh}$	w_{ha}	l_{ha}	λ_{hand}
com height	$\mathcal{T}_{cz}$	w_{cz}	l_{cz}	λ_{com}
com (x,y)	$\mathcal{T}_{cxy}$	w_{cxy}	l_{cxy}	λ_{com}
waist orientation	$\mathcal{T}_{wo}$	w_{wo}	l_{wo}	σ_{waist}
waist height	$\mathcal{T}_{wh}$	w_{wh}	l_{wh}	λ_{waist}
head orientation	$\mathcal{T}_{h}$	w_{h}	l_{h}	σ_{head}
chest orientation	$\mathcal{T}_{c}$	w_{c}	l_{c}	σ_{chest}
neck posture	$\mathcal{T}_{n}$	w_{n}	l_{n}	$\mu_{posture}$
torso posture	$\mathcal{T}_{t}$	w_{t}	l_{t}	$\mu_{posture}$
left arm posture	$\mathcal{T}_{la}$	w_{a}	l_{a}	$\mu_{posture}$
right arm posture	$\mathcal{T}_{ra}$	w_{a}	l_{a}	$\mu_{posture}$
left lower arm posture	$\mathcal{T}_{lla}$	w_{la}	l_{la}	$\mu_{posture}$
right lower arm posture	$\mathcal{T}_{rla}$	w_{la}	l_{la}	$\mu_{posture}$
left leg posture	$\mathcal{T}_{ll}$	w_{l}	l_{l}	$\mu_{posture}$
right leg posture	$\mathcal{T}_{rl}$	w_{l}	l_{l}	$\mu_{posture}$

Controller	**Considered tasks**
C1	$\mathcal{T}_t$ $\mathcal{T}_{lla}$ $\mathcal{T}_{rla}$ $\mathcal{T}_n$ $\mathcal{T}_{cxy}$ $\mathcal{T}_{cz}$ $\mathcal{T}_{wo}$ $\mathcal{T}_{lf}$ $\mathcal{T}_{rf}$ $\mathcal{T}_{lh}$ $\mathcal{T}_{rh}$ $\mathcal{T}_h$
C2	$\mathcal{T}_t$ $\mathcal{T}_{la}$ $\mathcal{T}_{la}$ $\mathcal{T}_{ll}$ $\mathcal{T}_{rl}$ $\mathcal{T}_n$ $\mathcal{T}_{cxy}$ $\mathcal{T}_{wo}$ $\mathcal{T}_{wh}$ $\mathcal{T}_h$ $\mathcal{T}_{lf}$ $\mathcal{T}_{rf}$

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 $\boldsymbol{b}_k$. 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 $\mathcal{T}_k$ considered for the controller are those for which the motion generator produces the references $\boldsymbol{b}_k$.

B. Control Configuration – parameters to optimize

For each task $\mathcal{T}_k$, we want to learn its Soft Priority Weight (SPW) w_k, its Convergence Gains (CG) (position error gain λ_k and orientation error gain σ_k for Cartesian tasks, postural error gain μ_k for joint tasks) and its position in the stack/hierarchy. A specific task $\mathcal{T}_k$ can be either active in one level $\mathcal{S}_i$ of the stack $\mathcal{S}$ 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 $\mathcal{S}$ as follows[2]:

[2]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.

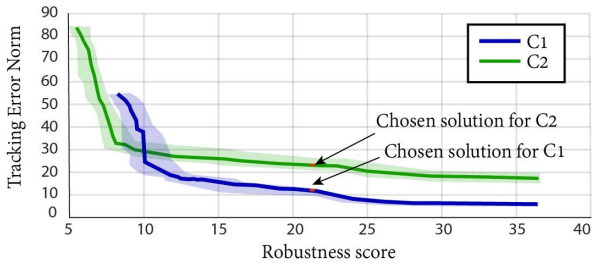

Fig. 3: Median Pareto front (thick line) and associated IQR (colored region) computed by NSGA-II in 20 runs with a population of 100 individuals and 300 generations for the controllers $C1$ (blue) and $C2$ (green).

$$\begin{cases} \mathcal{T}_k \subseteq \mathcal{S}_1 & \text{if } (0 \leq l_k \leq 0.25) \\ \mathcal{T}_k \subseteq \mathcal{S}_2 & \text{if } (0.25 < l_k \leq 0.5) \\ \mathcal{T}_k \subseteq \mathcal{S}_3 & \text{if } (0.5 < l_k \leq 0.75) \\ \mathcal{T}_k \; deactivated & \text{if } (0.75 < l_k \leq 1) \end{cases} \tag{4}$$

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

later on. The goal of the algorithm is to minimize f_1 and f_2 while maximizing f_3

$$\text{Minimize} \quad (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), -f_3(\boldsymbol{x})),$$

where $\boldsymbol{x}$ 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:

$$\begin{cases} f_{1_i} = f_{1_{i-1}} + (\sum_k \left|\Phi_i^k - \overline{\Phi}_i^k\right|) \\ f_{2_i} = f_{2_{i-1}} + x_{SP} + y_{SP} \quad \text{if (robot fallen)} \\ f_{2_i} = f_{2_{i-1}} + |x_i^{cz}| + |y_i^{cz}| \quad \text{if (robot not fallen)} \\ f_{3_i} = f_{3_{i-1}} - \alpha_3 \quad \text{if (robot fallen)} \end{cases} \tag{5}$$

where Φ_i^k and $\overline{\Phi}_i^k$ are respectively the reference and the actual value measured on the simulated robot of task k at time step i; x_i^{cz} is the distance in the frontal direction of the ZMP from the center of the SP at time step i, while y_i^{cz} 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 α_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 cm; for postural and orientation tasks, we consider for each angle the arc length in 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:

$$\begin{cases} f_1 = f_{1_{i-1}} + \alpha_{F1}(I - i) \quad \text{if } (i \neq 0) \\ f_2 = f_{2_{i-1}} + \alpha_{F2}(I - i) \quad \text{if } (i \neq 0) \\ f_3 = f_{3_{i-1}} - \alpha_{F3}(I - i) \quad \text{if } (i \neq 0) \\ f_1 = F_{max} \quad \text{if } (i = 0) \\ f_2 = F_{max} \quad \text{if } (i = 0) \\ f_3 = -F_{max} \quad \text{if } (i = 0) \end{cases} \tag{6}$$

where I is the total number of steps in the recorded sequence, i is the time step when the solver fails, F_{max} is the largest value a floating-point variable can hold, α_{F3} can be set to any arbitrary value greater than α_3 and lower than F_{max}/I and α_{F1}, α_{F2} to arbitrary values that are lower than F_{max}/I and respectively greater than the largest increment (Δ_1, Δ_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 α_{F1}, α_{F2}, 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 α_{F1}, α_{F2}. If the first generations indeed have large f_1, f_2 scores with α_{F1}, α_{F2} lower than Δ_1, Δ_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 $\alpha_{F1}, \alpha_{F2}, \alpha_{F3}, \alpha_3$ can be set to arbitrary values comprised in between lower and upper bounds that are known, with the exception of the lower bounds of α_{F1}, α_{F2} 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[3].

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 II). 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.

simultaneously rotate and incline its torso while shifting its weight on the right foot and moving the arms.

Controllers $C1$ and $C2$ 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 $C1$ and $C2$. They are specified by the set of tasks considered for their control configuration, reported in TABLE I. $C1$ takes as references (from the motion generator) mainly Cartesian tasks, while $C2$ mostly postural trajectories. Both controllers can be used by the humanoid to realize double support motions, but $C2$ 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 $S1, S2$ and $S3$, but only $C2$ will be further validated for the robot teleoperation.

Optimization on training sequences (offline): The parameters of the two controllers $C1$ and $C2$ are learnt by the algorithm NSGA-II implemented in Sferes$_{v2}$ [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[4]. 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 $\alpha_{F1}, \alpha_{F2}, \alpha_{F3} = 0.5$ 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 3).

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 $C1$ and $C2$ on the real robot. We use the three test sequences $S1, S2, S3$. 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 $C1$ with a squat motion with straight torso reference, that these solutions are not robust enough to be transferred onto the real robot, which falls. 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 $C1$ that represents the most frequent solution, given $f_2 = 22$ (see TABLE IV):

$$\begin{aligned}\overline{\mathcal{S}}_{C1} = (w_f(\mathcal{T}_{lf} + \mathcal{T}_{rf}) + w_h\mathcal{T}_h)/ \\ (w_{cxy}\mathcal{T}_{cxy} + w_{cz}\mathcal{T}_{cz} + w_t\mathcal{T}_t + \\ + w_{wo}\mathcal{T}_{wo} + w_{ha}(\mathcal{T}_{lh} + \mathcal{T}_{rh}))/ \\ (w_{la}(\mathcal{T}_{lla} + \mathcal{T}_{rla}));\end{aligned} \tag{7}$$

while for $C2$ we have:

$$\begin{aligned}\overline{\mathcal{S}}_{C2} = (w_f(\mathcal{T}_{lf} + \mathcal{T}_{rf}) + w_o\mathcal{T}_{wo} + w_n\mathcal{T}_n)/ \\ (w_{cxy}\mathcal{T}_{cxy} + w_{wh}\mathcal{T}_{wh} + w_t\mathcal{T}_t + w_a(\mathcal{T}_{la} + \mathcal{T}_{ra}))/ \\ (w_c\mathcal{T}_c);\end{aligned} \tag{8}$$

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[5].

Comparison with a baseline hand-tuned controller: We compare the performance of the learned controller for $C1$ with respect to a baseline solution that was manually tuned by an expert. The test is performed on the three different sequences $S1, S2, S3$.

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

$$\begin{aligned}\mathcal{S}_{HT} = (w_{ht}(\mathcal{T}_{lf} + \mathcal{T}_{rf}) + \mathcal{T}_{cxy} + \mathcal{T}_h)/ \\ (\mathcal{T}_{cz} + \mathcal{T}_t + (\mathcal{T}_{lh} + \mathcal{T}_{rh}))/ \\ ((\mathcal{T}_{lla} + \mathcal{T}_{rla}));\end{aligned} \tag{9}$$

where the the tasks concerning the feet contacts $\mathcal{T}_{lf}, \mathcal{T}_{rf}$

TABLE II: Convergence Gains (CG) associated to the 20 learned configurations given $f_2 = 22$

C1			**C2**		
CG	Median	IQR	CG	Median	IQR
λ_{hand}	0.0191	0.033	λ_{waist}	0.5491	0.3791
λ_{feet}	0.0577	0.064	λ_{feet}	0.2486	0.0983
σ_{feet}	0.0051	0.0059	σ_{feet}	0.0983	0.1231
λ_{com}	0.4426	0.1664	λ_{com}	0.5911	0.1946
σ_{waist}	0.0591	0.042	σ_{waist}	0.0652	0.0472
σ_{head}	0.0796	0.0234	σ_{head}	0.2778	0.2560
$\mu_{posture}$	0.5605	0.2145	$\mu_{posture}$	0.5162	0.0821
			σ_{chest}	0.6052	0.4009

TABLE III: Soft Priority Weights (SPW) associated to the 20 learned configurations given $f_2 = 22$

C1			**C2**		
SFW	Median	IQR	SFW	Median	IQR
w_{ha}	0.639	0.2131	w_a	0.8406	0.296
w_f	0.5357	0.1786	w_f	0.5835	0.2144
w_{cxy}	0.8368	0.1941	w_{cxy}	0.9519	0.1984
w_{wo}	0.8674	0.4832	w_{wo}	0.1613	0.2337
w_h	0.3343	0.3101	w_h	0.5357	0.2465
w_n	0.3256	0.2893	w_n	0.406	0.2419
w_t	0.9258	0.245	w_t	0.0656	0.2138
w_{la}	0.3599	0.3133	w_l	0.1145	0.3756
w_{cz}	0.7684	0.2191	w_{wh}	0.1879	0.1785
			w_c	0.9902	0.091

[4]Note that the optimization is not trajectory-specific and is only run once to get a controller that can achieve many trajectories.

[5]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.

TABLE IV: Task frequency in each level of the hierarchy in the 20 learned configurations, given $f_2 = 22$. (Most frequent solution in bold)

C1				C2			
Task	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	Task	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$
$\mathcal{T}_{lh},\ \mathcal{T}_{rh}$	2	**18**	0	$\mathcal{T}_{la},\ \mathcal{T}_{ra}$	3	**17**	0
$\mathcal{T}_{lf},\ \mathcal{T}_{rf}$	**18**	2	0	$\mathcal{T}_{lf},\ \mathcal{T}_{rf}$	**19**	1	0
$\mathcal{T}_{cxy}$	4	**16**	0	$\mathcal{T}_{cxy}$	3	**17**	0
$\mathcal{T}_{wo}$	6	**8**	0	$\mathcal{T}_{wo}$	6	**14**	0
$\mathcal{T}_{h}$	**11**	0	1	$\mathcal{T}_{h}$	6	1	0
$\mathcal{T}_{n}$	9	1	1	$\mathcal{T}_{n}$	**9**	4	0
$\mathcal{T}_{t}$	3	**17**	0	$\mathcal{T}_{t}$	1	**17**	2
$\mathcal{T}_{lla},\ \mathcal{T}_{rla}$	1	3	**16**	$\mathcal{T}_{ll},\ \mathcal{T}_{rl}$	0	0	5
$\mathcal{T}_{cz}$	4	**16**	0	$\mathcal{T}_{wh}$	3	**17**	0
				$\mathcal{T}_{c}$	0	1	**9**

and the center of mass $\mathcal{T}_{cxy}$ 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 λ_{feet}, σ_{feet}, λ_{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 $C1$ 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.

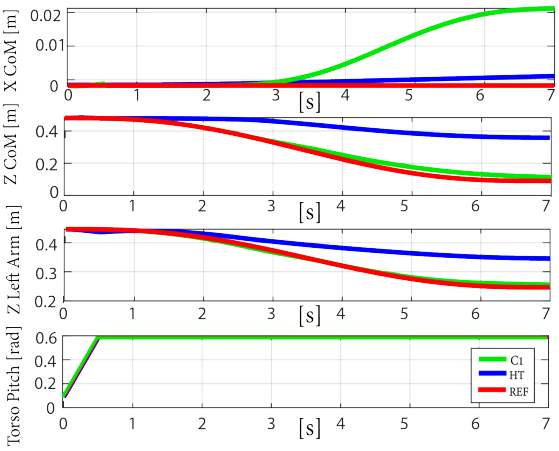

Fig. 4: Comparison of the tracking performance of the median learned controller $C1$ and the hand-tuned controller HT (with $w_{ht} = 1$) in sequence $S1$, on some significant tasks (in simulation).

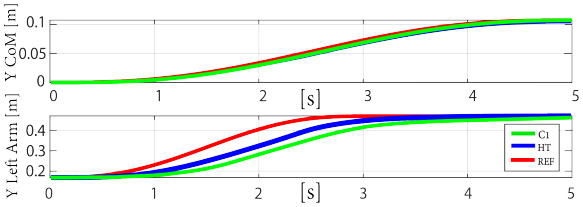

Fig. 5: Comparison of the tracking performance of the median learned controller $C1$ and the hand-tuned controller HT (with $w_{ht} = 0.7$, since the controller cannot find a solution for $w_{ht} = 1$) in sequence $S2$, on some significant tasks (in simulation).

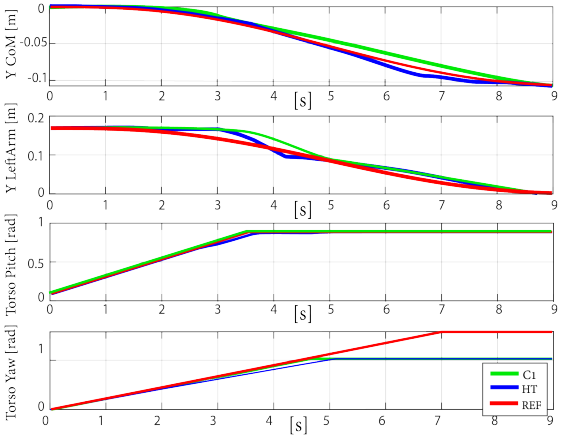

Fig. 6: Comparison of the tracking performance of the median learned controller $C1$ and the hand-tuned controller HT (with the same task priorities of $C1$) in sequence $S3$, on some significant tasks (in simulation).

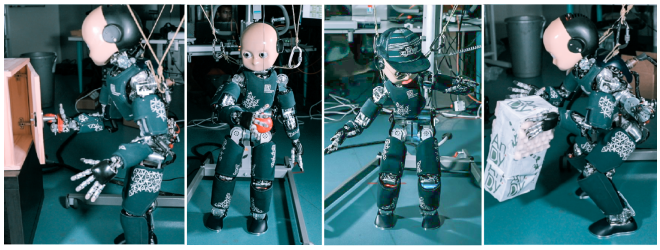

Fig. 7: The iCub robot controlled with the learned configuration $C2$ while performing different teleoperated tasks.

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 $C1$ 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

[1] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis, "Opensot: A whole-body control library for the compliant humanoid robot coman," in *2015 ICRA*, May 2015, pp. 6248–6253.

[2] A. Del Prete, F. Nori, G. Metta, and L. Natale, "Prioritized motion-force control of constrained fully-actuated robots:"task space inverse dynamics"," *Robotics and Autonomous Systems*, 10 2014.

[3] C. Ott, A. Dietrich, and A. Albu-Schäeffer, "An overview of null space projections for redundant, torque-controlled robots," *Automatica*, vol. 53, pp. 416–423, 03 2015.

[4] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, February 1987.

[5] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions," in *2011 ICRA*, May 2011, pp. 1283–1290.

[6] Y. T. M. Liu and V. Padois, "Generalized hierarchical control," in *Autonomous Robots*, 2016, pp. vol. 40, pp. 17–31.

[7] V. Modugno, G. Neumann, E. Rueckert, G. Oriolo, J. Peters, and S. Ivaldi, "Learning soft task priorities for control of redundant robots," in *ICRA*, May 2016, pp. 221–226.

[8] R. Lober, V. Padois, and O. Sigaud, "Variance modulated task prioritization in whole-body control," *IROS*, pp. 3944–3949, Sep. 2015.

[9] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann, "Probabilistic prioritization of movement primitives," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2294–2301, Oct 2017.

[10] N. Dehio, R. F. Reinhart, and J. J. Steil, "Multiple task optimization with a mixture of controllers for motion generation," in *IROS*, Sept 2015, pp. 6416–6421.

[11] F. Keith, P. Wieber, N. Mansard, and A. Kheddar, "Analysis of the discontinuities in prioritized tasks-space control under discreet task scheduling operations," in *IROS*, Sep. 2011, pp. 3887–3892.

[12] S. Kim, K. Jang, S. Park, Y. Lee, S. Yup Lee, and J. Park, "Continuous task transition approach for robot controller based on hierarchical quadratic programming," *IEEE RA-L*, 2019.

[13] N. Dehio and J. Steil, "Dynamically-consistent generalized hierarchical control," in *IROS*, May 2019.

[14] J. Silvério, S. Calinon, L. D. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations," *IEEE Transactions on Robotics*, vol. 35, pp. 78–94, 2019.

[15] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *CoRR*, vol. abs/1703.00472, 2017.

[16] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IROS*, Sept 2017, pp. 23–30.

[17] D. Clever et al., "Cocomopl: A novel approach for humanoid walking generation combining optimal control, movement primitives and learning and its transfer to the real robot HRP-2," *IEEE RAL*, vol. 2, no. 2, pp. 977–984, 2017.

[18] V. Modugno, U. Chervet, G. Oriolo, and S. Ivaldi, "Learning soft task priorities for safe control of humanoid robots with constrained stochastic optimization," in *HUMANOIDS*, 2016.

[19] A. Del Prete and N. Mansard, "Robustness to Joint-Torque Tracking Errors in Task-Space Inverse Dynamics," *IEEE Transaction on Robotics*, vol. 32, no. 5, pp. 1091 – 1105, 2016.

[20] M. Charbonneau, V. Modugno, F. Nori, G. Oriolo, D. Pucci, and S. Ivaldi, "Learning robust task priorities of QP-based whole-body torque-controllers," in *HUMANOIDS*, 2018.

[21] J. Carpentier et al., "Multi-contact Locomotion of Legged Robots in Complex Environments – The Loco3D project," in *RSS Workshop on Challenges in Dynamic Legged Locomotion*, Boston, July 2017, p. 3p.

[22] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *IJRR*, vol. 33, no. 7, pp. 1006–1028, 2014.

[23] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *IJRR*, no. 6, pp. 737–757, 2008.

[24] L. Penco et al., "Robust real-time whole-body motion retargeting from human to humanoid," *HUMANOIDS*, pp. 425–432, 2018.

[25] K. Deb and D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*. NY, USA: John Wiley & Sons, Inc., 2001.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[27] E. M. Hoffman, A. Laurenzi, L. Muratore, N. G. Tsagarakis, and D. G. Caldwell, "Multi-priority cartesian impedance control based on quadratic programming optimization," in *ICRA*, 2018, pp. 309–315.

[28] J. Lee et al., "Dart: Dynamic animation and robotics toolkit," *J. Open Source Software*, vol. 3, p. 500, 2018.

[29] J.-B. Mouret and S. Doncieux, "Sferesv2: Evolvin' in the multi-core world," in *IEEE Cong. Evolutionary Computation*, July 2010, pp. 1–8.

[30] L. Penco et al., "A multimode teleoperation framework for humanoid loco-manipulation: A demonstration using the icub robot," *IEEE RAM*, 2019.

[31] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Auton. Robots*, vol. 40, no. 3, p. 429–455, 2016.