Automatic Tuning and Selection of Whole-Body Controllers

D'Elia, E.; Mouret, J. -B.; Kober, J.; Ivaldi, S.

DOI
10.1109/IROS47612.2022.9981058

Publication date
2022

Document Version
Final published version

Published in
Proceedings 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

Automatic Tuning and Selection of Whole-Body Controllers

Evelyn D'Elia[1,2,3], Jean-Baptiste Mouret[1], Jens Kober[2], Serena Ivaldi[1]

***Abstract*— Designing controllers for complex robots such as humanoids is not an easy task. Often, researchers hand-tune controllers, but this is a time-consuming approach that yields a single controller which cannot generalize well to varied tasks. This work presents a method which uses the NSGA-II multi-objective optimization algorithm with various training trajectories to output a diverse Pareto set of well-functioning controller weights and gains. The best of these are shown to also work well on the real Talos robot. The learned Pareto front is then used in a Bayesian optimization (BO) algorithm both as a search space and as a source of prior information in the initial mean estimate. This combined learning approach, leveraging the two optimization methods together, finds a suitable parameter set for a new trajectory within 20 trials and outperforms both BO in the continuous parameter search space and random search along the precomputed Pareto front. The few trials required for this formulation of BO suggest that it could feasibly be applied on the physical robot using a Pareto front generated in simulation.**

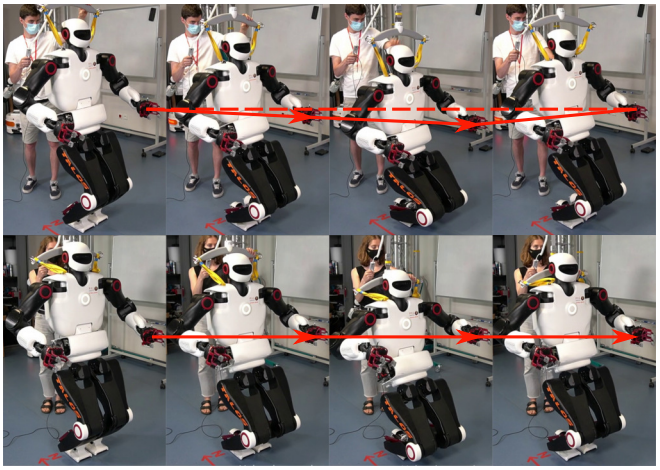

Fig. 1. Comparison of squat trajectory performance between hand-tuned control parameters (top) and parameters learned using our method (bottom), on the Talos robot.

I. INTRODUCTION

Despite recent advances in robotic control, researchers still encounter a myriad of difficulties when controlling humanoid robots. Ideally, a humanoid robot should have versatile capabilities and should be able to complete manual labor that is grueling or even unsafe for humans. However, since humanoid robots are extremely complex and are meant to be capable of performing vastly different trajectories such as locomotion or manipulation, it is difficult to design a stable, accurate controller, and it is also difficult to find controllers that are transferable. Our approach addresses both of these problems.

One way to solve this problem is by breaking the control down into a set of tasks of varying importance, each with a gain parameter to define its behavior. This is called a task priority-based whole-body control (WBC) approach, and can be implemented with either *strict priorities* [1], [2], [3], which ensure more important tasks are completed at the expense of less important ones, *soft priorities* [4], which allow a continuum of priority level, or both [5]. The priority-based formulation streamlines controller design, but still requires the user to choose task parameters that perform well. One typical way to choose these parameters is by hand-tuning them, but since this method is time-consuming, it often yields only one controller to be used for many trajectories. The main drawback of this method is that a single "robust" controller compromises the quality of individual trajectories.

Due to the inconvenience of hand-tuning WBC parameters, many recent approaches utilize learning algorithms to automatically tune them. Some such approaches leverage programming by demonstration to learn task parameters for manipulators [6], [7], [8]. The drawbacks of these methods are that they require time to carry out demonstrations. Conversely, an imitation learning approach proposed by Silvério et. al. [9] is implemented on the humanoid COMAN, using simulation-generated demonstrations, but validation of this method is limited to manipulation trajectories.

Instead of using demonstrations to guide learning, optimization algorithms can be used to find suitable task parameters. In most cases of learning on a real robot, it is beneficial to first optimize controllers in simulation. One class of optimization algorithms, evolutionary algorithms, are useful in robotics when the search is in continuous space and gradient-based optimization is impossible. One well-known and often used evolutionary method is CMA-ES [10]. The authors of [11], [12] and [13] employ CMA-ES to search for optimal trajectory parameters in the form of basis function weights for simulated tasks on the child-sized humanoid iCub [14]. The approach of [12] uses a two-step optimization, the first of which is unconstrained CMA-ES to find a feasible starting point for the second step. On the other hand, authors of [11] initialize with hand-tuned parameters, while those of [13] skip the bootstrapping step by using retargeted recorded

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 731540 (project AnDy) and No. 101070596 (project euROBIN) and partly funded by the CPER project "CyberEntreprises" and the CPER project SCIARAT.

[1]Inria, University of Lorraine, CNRS, Loria, Nancy, France
[2]Delft University of Technology, Delft, Netherlands
[3]Artificial and Mechanical Intelligence Lab, Italian Institute of Technology, Genoa, Italy evelyn.delia@iit.it

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-7927-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IROS47612.2022.9981058

human trajectories to initialize the optimization problem. Both of these studies highlight an important weakness of CMA-ES: it must be initialized within the feasible region to find a solution. Also, as a single-objective method, it cannot individually optimize multiple goals at once.

Another evolutionary algorithm, Non-dominated Sorting Genetic Algorithm II (NSGA-II) [15], a multi-objective optimization (MOO) algorithm which solves the issues associated with CMA-ES, is employed by Penco et al. [16] to create a set of simulation-trained Pareto-optimal controllers, which are then used to decrease the number of real-robot tests required. However, since this method trades off only two objectives, accuracy and stability, the results lack diversity. In this paper we aim to use NSGA-II with one objective per training trajectory, to allow the final set of controllers to be more varied.

Using a pre-generated set of possible solutions as the search space, a single-objective learning algorithm can be used to transfer Pareto solutions to new trajectories that are not part of the MOO training set, and choosing one that is data-efficient, as opposed to a big-data deep learning approach, leaves open the possibility of applying the approach on the real robot. One data-efficient class of learning algorithms, called policy search (PS) [17], [18], is very promising in the context of real-world robot learning as a method that learns a successful controller with as little data as possible. In particular, the Bayesian optimization (BO) algorithm requires fewer trials than other PS methods [18].

Prior information has the potential to speed up learning drastically. Some recent work with BO encodes prior knowledge within the Gaussian process (GP) kernel function. For example, the Behavior-Based Kernel (BBK) introduced in [19] encodes information about policy closeness by calculating the Kullback-Leibler divergence. Antonova et al. [20], [21] designed two kernels: one called Determinants of Gait (DoG), which has 16 hand-picked policy parameters to describe a gait, and uses short simulations to evaluate the success of a given parameter vector, and another called trajNN, which is learned with a neural network. Learning with these more informative kernels does improve the speed of learning, but they also make the algorithm much more complex.

Instead of incorporating them in the kernel, Cully et al. [22] encode priors into the GP mean function by first precomputing a behavior-performance map, which is a map in parameter space of how well the robot performs in simulation, and then using this map as the GP mean initial guess, which is shown to guide the real-world BO toward likely high-performing solutions.

We propose a two-pronged solution to the aforementioned challenges: a multi-objective optimization (MOO) step to generate a large set of control parameter options, and a Bayesian optimization step to narrow down these solutions to one which works for a new, untrained trajectory or a new robot model. The results of this approach surprisingly show that between different simulated robot models, the Pareto fronts generated by NSGA-II perform similarly, which indicates that there may be no need to use BO for closing the reality gap. This is why we focus instead on achieving transfer to novel trajectories.

II. BACKGROUND

A. Task priority-based control

The control framework we employ is a generic whole-body control (WBC) formulation known as task priority-based control [1], with *soft task priorities*. Here a task is defined, depending on its type, as either a Cartesian position and/or orientation, or a set of joint angles, to track. Task priority-based control solves the following QP optimization problem at each time step to choose the next control input:

$$\begin{aligned} \min_{\boldsymbol{u},\ddot{\boldsymbol{q}}} \quad & \sum_{i=0}^{n_t} w_i \left\| \boldsymbol{A}_i(\boldsymbol{q},\dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} - \boldsymbol{b}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) \right\|^2 + \epsilon \left\| \boldsymbol{u} \right\|^2 \\ \text{s.t.} \quad & \begin{cases} \boldsymbol{A}_{i,ineq}(\boldsymbol{q},\dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} \leq \boldsymbol{b}_{i,ineq}(\boldsymbol{q},\dot{\boldsymbol{q}}) \\ \boldsymbol{A}_{i,eq}(\boldsymbol{q},\dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} = \boldsymbol{b}_{i,eq}(\boldsymbol{q},\dot{\boldsymbol{q}}) \\ \boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{J}(\boldsymbol{q})^\top \boldsymbol{f}, \end{cases} \end{aligned} \tag{1}$$

where $\boldsymbol{u}$ is the control input torque, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the joint positions, velocities, and accelerations, n_t is the number of tasks, w_i is the soft priority weight (SPW), $\boldsymbol{A}_i$ is the task's equivalent Jacobian, $\boldsymbol{b}_i$ is the reference, ϵ is a regularization factor, and $\boldsymbol{f}$ is the external wrench. The constraints represent all control input, dynamics, environmental, and most importantly, model constraints on the robot. The reference $\boldsymbol{b}_i$, which depends on the desired trajectory, is defined for both Cartesian and postural tasks as:

$$\boldsymbol{b}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) = \ddot{\boldsymbol{p}}_i^d - \dot{\boldsymbol{A}}_i(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \lambda_i^P \boldsymbol{e} + \lambda_i^D \dot{\boldsymbol{e}}, \tag{2}$$

where superscript d refers to desired value, $\ddot{\boldsymbol{p}}_i^d$ is desired task acceleration, $\dot{\boldsymbol{A}}_i$ is the time derivative of the task Jacobian, $\boldsymbol{e} = \boldsymbol{p}_i - \boldsymbol{p}_i^d$, $\dot{\boldsymbol{e}} = \dot{\boldsymbol{p}}_i - \dot{\boldsymbol{p}}_i^d$ are the errors between desired and actual position $\boldsymbol{p}_i$ and velocity $\dot{\boldsymbol{p}}_i$ vectors, where the goal position and/or orientation depends on the task i, and λ_i^P and $\lambda_i^D = 2\sqrt{\lambda_i^P}$ are the proportional and derivative task convergence gains (CG), according to [23].

B. NSGA-II

The first part of the approach outlined here employs one of the most popular MOO algorithms, NSGA-II. NSGA-II is part of a class of evolutionary algorithms, whose strength lies in their ability to find a diverse set of Pareto optimal solutions, an important characteristic for finding a robot controller that is able to generalize to many types of tasks. NSGA-II outperforms other evolutionary MOO methods in terms of the diversity and accuracy of solutions found [15].

MOO is a subset within the optimization field that allows optimization to be done over multiple objective functions, instead of a single one. The problem is formulated as:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \{f_1(\boldsymbol{\theta}), f_2(\boldsymbol{\theta}), \ldots, f_l(\boldsymbol{\theta})\} \\ \text{s.t.} \quad & \boldsymbol{\theta} \in S, \end{aligned} \tag{3}$$

where $f_n(\boldsymbol{\theta})$ is an objective function, l is the number of objectives and S is the feasible set to which the parameter vector $\boldsymbol{\theta}$ must belong. Since the algorithm optimizes

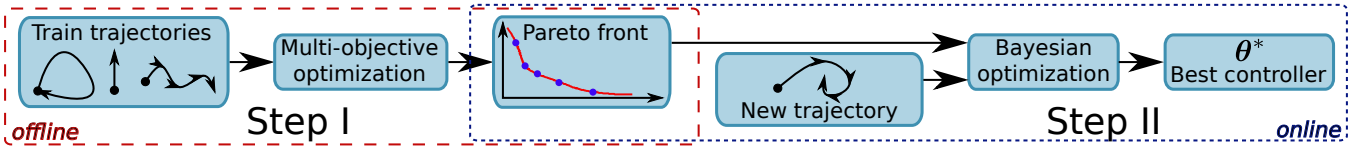

Fig. 2. Outline of our two-part approach. Step I uses a multi-objective optimization method, NSGA-II, to train a Pareto front of controller solutions on various whole-body trajectories. This step is performed offline, on the simulated robot. Step II uses this Pareto front as a search space with Bayesian optimization and chooses the best-performing controller in the front for a new trajectory. This step is performed online, on the real robot.

many objective functions at once, without aggregating them together, there seldom exists a single solution (a point in parameter space within the feasible set S) that optimizes every objective function. Here, the concept of *Pareto dominance* [24] is used: A solution $\boldsymbol{\theta_1}$ dominates another solution $\boldsymbol{\theta_2}$, if and only if the following conditions are satisfied: (i) $\boldsymbol{\theta_1}$ is not worse than $\boldsymbol{\theta_2}$ with respect to all objectives, and (ii) $\boldsymbol{\theta_1}$ is strictly better than $\boldsymbol{\theta_2}$ with respect to at least one objective. A *Pareto optimal* solution is a solution for which no one objective function can be further optimized without compromising the performance of another objective, i.e., it is non-dominated. Solving a MOO problem results in a set of Pareto optimal solutions, called the Pareto front, which each perform differently with respect to each of the various objective functions.

C. Bayesian optimization

The second portion of this work aims to facilitate transfer of Pareto solutions to the real robot. For this, a surrogate-based PS method [18], Bayesian optimization, is chosen because this type of algorithm conforms to this particular problem's need for few rollouts.

BO is distinguished by the fact that it learns a surrogate model of the expected return $\hat{J}(\boldsymbol{\theta})$, most often represented by a GP:

$$\hat{J}(\boldsymbol{\theta}) \sim \mathcal{GP}(\mu(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}')), \tag{4}$$

where $\mu(\boldsymbol{\theta})$ is the GP mean function, and $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$ is the covariance, or kernel function. Using the set of recorded returns from past rollouts $D_{1:t}$, a distribution of the expected return can be estimated for a new parameter set $\boldsymbol{\theta}_*$ using the predictions of the GP:

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= \boldsymbol{k}^\top K^{-1} D_{1:t} \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^\top K^{-1} \boldsymbol{k}, \end{aligned} \tag{5}$$

where $\boldsymbol{k} = k(D_{1:t}, \boldsymbol{\theta}_*)$ is a kernel vector, K is a kernel matrix with $K_{i,j} = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$, $\mu(\boldsymbol{\theta}_*)$ is the mean prediction of the GP and $\sigma^2(\boldsymbol{\theta}_*)$ is the GP variance prediction.

The kernel function $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, used to quantify the difference in expected return $\hat{J}(\boldsymbol{\theta})$ between two possible parameter sets, can be defined in various ways to shape the performance of the optimization.

In BO, the acquisition function, a measure of how much information will be provided by a parameter set, uses the surrogate model $\hat{J}(\boldsymbol{\theta})$ to choose which policy parameters to test next, balancing exploration and exploitation. Upper Confidence Bound (UCB) [25], which is found to be the best of three common options [26], [27], optimistically judges each prospective parameter set by the upper limit of the GP of its expected return:

$$\alpha_{UCB}(\boldsymbol{\theta}; D_{1:t}) = \mu(\boldsymbol{\theta}) + \beta\sigma(\boldsymbol{\theta}), \tag{6}$$

where β is a hyperparameter that balances exploitation and exploration by specifying how high the upper confidence is.

III. METHODS

Our method has two overall steps, shown in Figure 2: optimize a Pareto front using the evolutionary MOO algorithm, NSGA-II, and then use that Pareto front as prior information to achieve solution transferability with BO.

A. Task formulation

In this work the control parameter set is composed of task priority-based control weights and gains as described in Section II-A. We optimize two parameters per task: the SPW w_i, which measures the importance of the task compared to the other tasks, and the CG λ_i^P, which defines the responsiveness of the controller to errors in task position and velocity. We relate the derivative CG λ_i^D directly to λ_i^P and thus it is not independently optimized. The parameter set $\boldsymbol{\theta}$ consists of these two parameter values, w_i and λ_i^P, for each task.

B. Pareto front dimensions

To generate Pareto fronts with NSGA-II, we measure the quality of a given parameter set using objective functions. The main aspect of our NSGA-II implementation that sets it apart from that of [16] is the objective function setup that is used. Penco et al. [16] use two objective functions, where the performance of each of these is averaged out over the training trajectories. Since different movement types necessitate unique control parameters, we instead use a separate objective function for each training trajectory chosen in order to ensure diversity in the learned solutions. This way the resultant Pareto front is l-dimensional, where l is the number of training trajectories used.

C. Inclusion of prior information

Our implementation of Pareto-based BO uses the previously generated Pareto front as a search space, making the parameter space finite and easily searchable. This initialization strategy is similar to that of [22], where the initialization contained prior information in the form of a behavior-performance map that was pre-computed in simulation. Our BO method differs from that approach in the way that the

TABLE I

SYMBOL, DESCRIPTION, SPW NAME AND CG TYPE FOR EACH OPTIMIZED TASK.

Task	Description	SPW	CG
$\mathcal{T}_{\{rh,lh\}}$	hand pose (symmetric)	w_h	$\lambda_h^P = \sigma_h^P$
$\mathcal{T}_{\{rf,lf\}}$	foot pose (symmetric)	w_f	$\lambda_f^P = \sigma_f^P$
$\mathcal{T}_{CoM}$	CoM position	w_{CoM}	λ_{CoM}^P
$\mathcal{T}_{to}$	torso orientation (roll, pitch)	w_{to}	σ_{to}^P
$\mathcal{T}_p$	joint angles	w_p	μ_p^P

map (Pareto front in this case) is generated. Since in our case the parameter space is easily searchable, instead of using an optimizer, an exhaustive search algorithm similar to that used in [22] is employed. The mean function $\mu(\boldsymbol{\theta})$ of the GP, which has a significant effect on the algorithm performance, is initialized using the average of the costs from the NSGA-II training trajectories. We use the Matérn $5/2$ kernel function to represent the GP covariance $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, and the UCB acquisition function to choose the next parameter set to test.

IV. EXPERIMENTS

In these experiments, we used the Task Space Inverse Dynamics (TSID) library [28] on top of the Pinocchio framework [29] in C++ to perform inverse dynamics task priority-based control (described in Section II-A). The robot used in this work is the 32-DoF Talos humanoid from PAL Robotics [30], on which TSID runs at a frequency of 500 Hz.

A. NSGA-II: Experimental design

NSGA-II was parallelized on a 256-core computer using the Sferes$_{v2}$ evolutionary algorithm framework [31]. Parallelization drastically reduces the amount of time required for this computationally intensive algorithm, but nonetheless, to complete a single NSGA-II run of 100 generations with four objective functions and self-collision checking enabled requires roughly 24 hours to run.

1) Tasks used: As shown in Table I, there are five tasks used to guide the movements of the Talos. In total, the parameter set has 10 elements. Each of these was optimized in the range of $[0, 2000]$, the same range used when hand-tuning the task parameters.

2) Objective function: For our implementation, in both the NSGA-II step and the BO step, each objective function is a measure of the Cartesian accuracy of 3 selected tasks: $\mathcal{T}_{\{rh,lh\}}$, $\mathcal{T}_{\{rf,lf\}}$, and $\mathcal{T}_{CoM}$. The form of each of these objective functions is:

$$f_b = \begin{cases} \sum_{t=0}^{T} \frac{\left(\left\|\boldsymbol{e}_t^{CoM}\right\| + \left\|\boldsymbol{e}_t^{\{rh,lh\}}\right\| + \left\|\boldsymbol{e}_t^{\{rf,lf\}}\right\|\right)}{N_t}, & \text{if no fail} \\ 1.0 \times 10^{10}, & \text{if fail,} \end{cases} \tag{7}$$

where b is the training trajectory index, N_t is the total number of time steps, T is the final time, t is the current time, and $\boldsymbol{e}$ is the 3-D Cartesian position (without orientation) error of the specific task at time t. The robot can fail either by falling over or by self-colliding. Using a separate objective function per training trajectory ensures that the learned Pareto front will contain a diverse set of parameters, likely to work well on a variety of test trajectories.

However, during trials on the real robot, due to the limitations of motion capture, it was not possible to accurately record the CoM position. Instead, a modified version of the objective function, which calculates the average tracking error of only the hands and feet, was used to compare each of the results:

$$f_{b,mod} = \frac{1}{N_t} \sum_{t=0}^{T} \left(\left\|\boldsymbol{e}_t^{\{rh,lh\}}\right\| + \left\|\boldsymbol{e}_t^{\{rf,lf\}}\right\| \right), \tag{8}$$

Equation (8) was used to compute the cost of each real robot experiment from the motion capture measurements.

3) Training trajectories: The more diverse the Pareto front, the more likely it is to transfer well to new trajectories. Since the trajectories used by NSGA-II to learn the Pareto front have a significant effect on this diversity, it is essential to choose training trajectories that use different parts of the robot and have different goals.

In all, there are 8 training trajectories used, each of which has a duration of 20 s. These 8 trajectories are evenly balanced with 4 handmade trajectories (walk on spot, squat, clap, and touch ground), for which the target positions were chosen by hand, and 4 trajectories retargeted from human motions recorded with the XSens MVN motion tracking suit [32] (dance, lean and twist, right arm reach, and lift). The result of combining these trajectories is a suitably varied and diverse training set. The full training set was divided in half (Set 1: walk on spot, clap, lean and twist, right arm reach, Set 2: squat, touch ground, dance, lift) and NSGA-II was run with 4 training trajectories at a time, for 100 generations each. Five Pareto datasets were optimized each for Set 1 and 2. Each set contains 2 handmade and 2 recorded trajectories.

4) Modified robot models: In order to evaluate the *transferability* (i.e., the success in crossing the reality gap) of the Pareto solutions from NSGA-II, 5 different mass-modified models (URDF files) of the Talos robot were created: uniformly 10% heavier and 10% lighter, and with 5 kg added to the left shoulder, right hand, and as a backpack.

5) Experiments on the real Talos: The learned parameters we chose to test on the real Talos were the best non-failing solutions in the Gazebo simulator, which has higher fidelity than the simulator used for NSGA-II optimization. The values of the hand-tuned parameters used and the chosen NSGA-II-generated parameter sets are listed in Table II.

The 3-D hand and foot tracking data from the motion capture system was compared to the reference positions sent to the robot over the course of each rollout. Here, Umeyama's method [33] was used to estimate the proper transformation.

B. NSGA-II: Results

1) Learned parameter values: The datasets resulting from our implementation of NSGA-II each contain roughly 100 Pareto solutions. For SPWs, Figure 3 shows that w_{CoM} and w_p are, respectively, very high and very low for almost every

TABLE II

HAND-TUNED AND LEARNED PARAMETER SETS USED FOR REAL ROBOT TESTING.

Trajectory	Type	w_h	w_f	w_{CoM}	w_{to}	w_p	λ_h^P	λ_f^P	λ_{CoM}^P	σ_{to}^P	μ_p^P
Squat	Hand-tuned	10	1000	1000	10	1.75	30	30	30	30	10
Squat	Learned	1296.6	1241.3	1871.3	248.9	26.1	1591.1	916.9	1968.6	618.0	231.5
Dance	Hand-tuned	100	100	2000	1000	100.75	1000	30	1000	30	60
Dance	Learned	207.9	192.1	1885.1	852.3	45.9	898.6	1074.7	1935.8	26.0	215.9

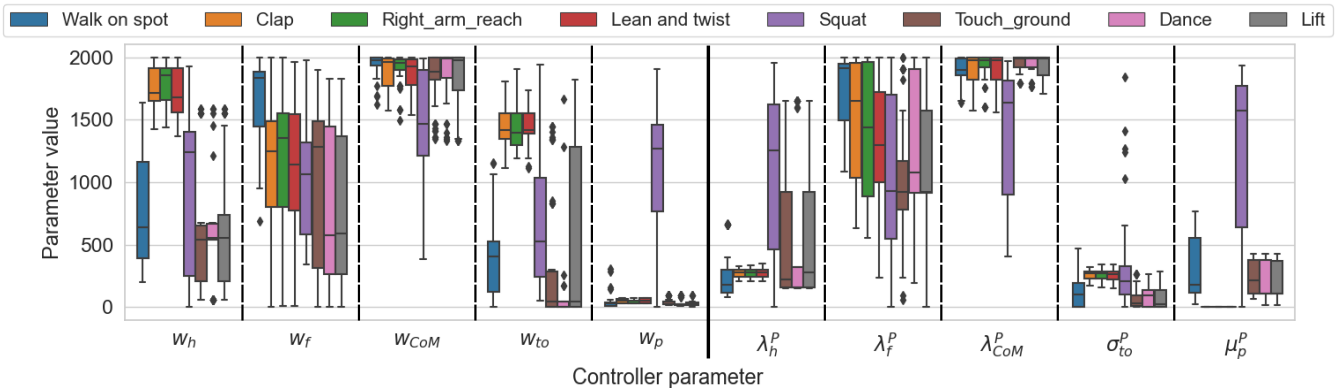

Fig. 3. Boxplot comparison of top 50 best-performing learned SPWs and CGs for each training trajectory. Data is from 5 separate Pareto fronts for each training set.

trajectory. Interestingly, the hand weights w_h for the clap, lean and twist, and right arm reach are very close to the upper bound of 2000, because the robot places high importance on the position of the hands to avoid collisions.

The CG boxplots in Figure 3 show that the λ_{CoM}^P values are universally very close to 2000, which is necessary to allow the robot to maintain stability. On the other hand, the torso and posture CGs are generally nearer to zero. It can be postulated that by making these robot parts less reactive, there are less jerky motions, lowering the risk of both self-collision and falling.

The squat is the most stable and therefore "easy" trajectory to find control parameters for, so for this reason it has a wider range of successful weights and gains. Conversely, λ_f^P values for each trajectory have a wide range and a relatively high median, suggesting that the specific value of this parameter does not have as large an influence on the possibility of failure, but that it does play a role in stabilizing the robot.

Additionally, Figure 3 shows that the actual mean parameter values learned for a robust controller are very different from those in the hand-tuned set, further illustrating the usefulness of optimization via this method.

2) Performance on modified models: One sample Pareto front each for training Sets 1 and 2 was tested on the 5 modified robot models described in Section IV-A.4. We found that between different models, the best-performing region of the Pareto front is the same, which suggests transfer from simulation to reality is straightforward.

3) Experiments on the real robot: After evaluating the performance of the learned Pareto fronts in simulation, the squat and dance trajectories were validated on the real Talos robot to compare the performance with learned vs. hand-tuned parameters. A video comparison of the learned and hand-tuned performance of these trajectories on the Talos is attached[1].

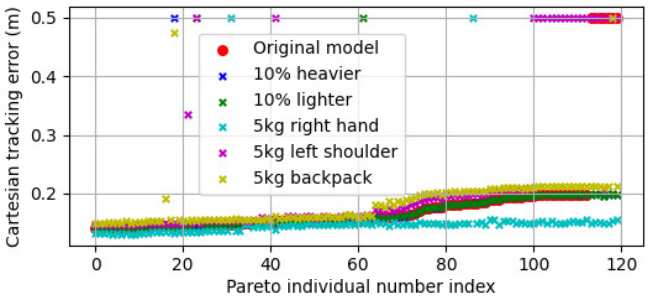

Fig. 4. Examples of the change in performance of a Pareto front between one robot model and another, for the walk on spot trajectory. Solutions with tracking error greater than 0.5 are labeled as failing solutions.

For the squat trajectory, Figure 1 shows that the hands move much less in the world frame with learned vs. hand-tuned parameters, which makes the whole movement more stable and precise. For the dance trajectory, it is more difficult to visualize the difference of the two solutions, because the robot is moving quite fast; however, the calculated cost of 8.96 cm for the learned parameter set is about 3 cm less than the 11.79 cm of the hand-tuned set, which is encouraging for a single trial.

In summary, both Pareto solutions that were tested worked well on the real robot. From these results we conclude that in addition to producing control parameters much more efficiently than hand-tuning, learning in simulation via NSGA-II also yields parameter sets that are transferable to reality.

[1]Video also available at: https://youtu.be/X1iNwDKXUNQ

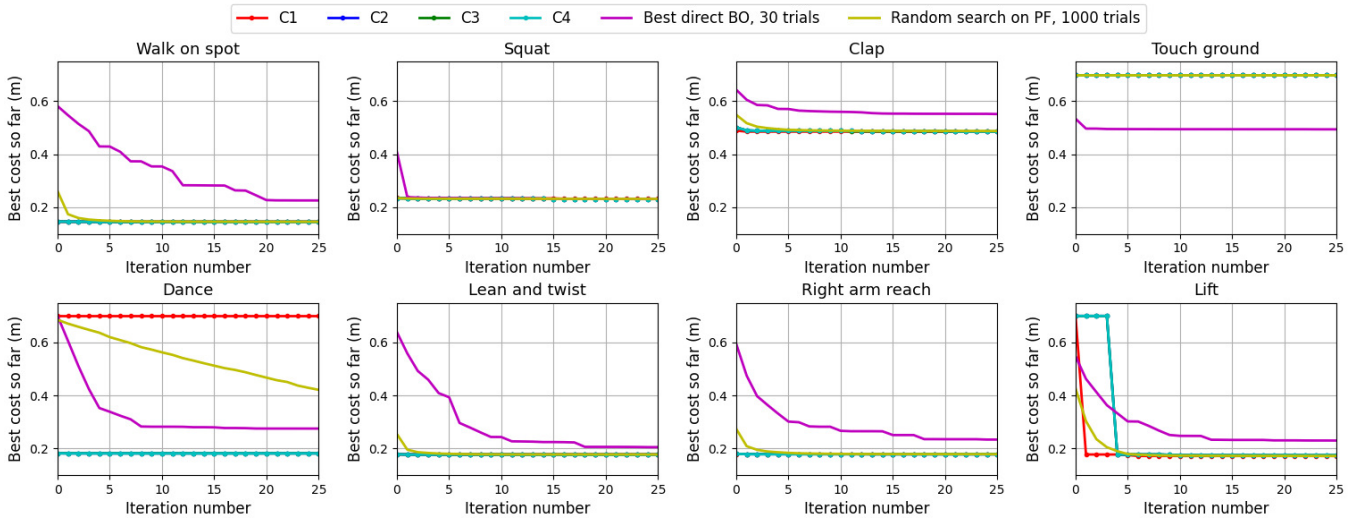

Fig. 5. Performance of our method, Pareto-based BO, on hyperparameter combinations C1-4 (deterministic, run once each) from Table III, compared to direct BO and random search (along the Pareto front) results. A cost of 0.7 is used to represent all failing costs. The different combinations perform similarly, but uniformly better than either direct BO or random search.

TABLE III
LIST OF SETTING COMBINATIONS TESTED FOR BO.

Combination name	$\mu_0(\boldsymbol{\theta})$	l_b	β
C1	mean w/ failing costs	1.0	0.5
C2	mean w/out failing costs	1.0	0.5
C3	mean w/out failing costs	5.0	0.5
C4	mean w/out failing costs	1.0	2.0

C. BO: Experimental design

BO was tested along the Pareto fronts generated from a sample MOO dataset to allow the robot to learn to follow new trajectories, on which the Pareto front was not trained. For this, each Set 1 trajectory was used as a "new" trajectory on the Set 2 Pareto front, and vice versa.

1) Algorithm settings: For Bayesian optimization, the many possible hyperparameter settings of the algorithm affect its performance differently. We chose to vary the initial GP mean function, the GP kernel smoothness l_b, and the UCB exploration coefficient β. The choice of initial mean $\mu_0(\boldsymbol{\theta})$ determines whether the optimizer is optimistic or pessimistic, which influences its ability to find solutions.

Since Pareto front-based BO has access to the Pareto solutions, an optimistic initial mean is an average of the NSGA-II-trained fitness values over all training trajectories for each individual, ignoring failing costs. In contrast, the pessimistic initial mean for Pareto-based BO is the same average, but includes failing costs, which bring down the average significantly. Next, changing the smoothness hyperparameter l_b affects how much the estimated expected returns $\hat{J}(\boldsymbol{\theta})$ of parameter sets near the sampled one are changed. The last part that will be examined is the exploration coefficient $\beta > 0$ of the UCB acquisition function, which balances exploration and exploitation. In order to evaluate which of these settings work best, each of the combinations C1-4, listed in Table III, are tested.

D. BO: Results

Using BO allows us to train a Pareto front on a few trajectories, but later have a means of adding new trajectories to the robot's repertoire without having to repeat the MOO.

In addition to testing this approach on each of our training trajectories, we also perform trials with the hyperparameter combinations given in Table III. In order to gain a better understanding of how well this method performs, Figure 5 compares it both to direct BO, which searches in the continuous space, and random search of the Pareto front space, averaged out over 1000 runs. Figure 5 indicates that in most situations, the Pareto-based BO approach finds very good control solutions within the first few iterations, suggesting that this approach could also be applied on a real robot. Our approach outperforms both direct BO and Pareto-based random search in terms of convergence time and solution quality.

1) Pareto-based BO vs. benchmarks: One immediate takeaway from Figure 5 is that direct BO finds conspicuously worse solutions than either the random search along the Pareto front or Pareto-based BO. This proves that there is a clear benefit to using a Pareto front as a small, discrete search space. It should be noted that for the touch ground trajectory, none of the Pareto-based methods find a successful solution. In fact, no successful solution exists for this trajectory on the searched Pareto front. This points to either a lack of adequate diversity in the trajectory group which Set 1 is trained on, or an insufficient number of training generations with NSGA-II.

2) Comparison of different settings: C1-4 perform similarly, but uniformly better than either direct BO or random search. Notably, the filtered mean configurations outperform the unfiltered C1 for every trajectory except the clap. In general, the filtered mean describes the search space better than the unfiltered mean, which is full of large error estimates.

V. CONCLUSIONS AND FUTURE WORK

In this paper we present a solution for learning task-priority-based control parameters for a humanoid robot in two steps: optimizing a Pareto front of parameter sets with NSGA-II, and searching in that Pareto front to learn suitable parameters for new trajectories with BO. This combination of methods saves time that would otherwise be spent hand-tuning a controller, improves tracking accuracy on individual trajectories, and is transferable to new, untrained trajectories.

One direction which deserves further investigation is to design and test other objective functions. The objective function used in this work assigns the same cost to all failing solutions and therefore provides little information to the optimizer about which direction in which to move in the parameter space to avoid failure. A more descriptive objective function, such as one that accounts for how much of the trajectory the robot completes before failing, would speed up the optimization. Other ways of initializing the GP mean function could also be useful to test, such as by measuring which NSGA-II training trajectory is most similar to the BO trajectory via some heuristic, and then using that training trajectory's Pareto front performance as the initial mean for learning the new trajectory. Another improvement consists of using whole-body retargeting to provide more complex behaviors, as was done in [16] for the iCub robot.

To further investigate the transferability of the approach from simulation to reality, more tests on the real robot will help to statistically analyze the squat and dance performance as well as to validate performance on other trajectories.

REFERENCES

[1] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The Int. Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.

[2] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *IEEE Int. Conf. on Advanced Robotics*, 1991.

[3] L. Sentis and O. Khatib, "Control of free-floating humanoid robots through task prioritization," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2005.

[4] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.

[5] M. Liu, R. Lober, and V. Padois, "Whole-body hierarchical motion and force control for humanoid robots," *Autonomous Robots*, vol. 40, no. 3, pp. 493–504, oct 2015.

[6] L. Armesto, J. Bosga, V. Ivan, and S. Vijayakumar, "Efficient learning of constraints and generic null space policies," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[7] C. A. V. Perico, J. D. Schutter, and E. Aertbelien, "Combining imitation learning with constraint-based task specification and control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1892–1899, 2019.

[8] D. Mronga and F. Kirchner, "Learning context-adaptive task constraints for robotic manipulation," *Robotics and Autonomous Systems*, vol. 141, p. 103779, jul 2021.

[9] J. Silverio, S. Calinon, L. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 78–94, 2019.

[10] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[11] N. Dehio, R. F. Reinhart, and J. J. Steil, "Multiple task optimization with a mixture of controllers for motion generation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[12] V. Modugno, G. Nava, D. Pucci, F. Nori, G. Oriolo, and S. Ivaldi, "Safe trajectory optimization for whole-body motion of humanoids," in *IEEE-RAS Int. Conf. on Humanoid Robotics (Humanoids)*, 2017.

[13] W. Gomes, V. Radhakrishnan, L. Penco, V. Modugno, J.-B. Mouret, and S. Ivaldi, "Humanoid whole-body movement optimization from retargeted human motions," in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2019.

[14] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The iCub humanoid robot," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems - PerMIS '08*. ACM Press, 2008.

[15] 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, Apr 2002.

[16] L. Penco, E. M. Hoffman, V. Modugno, W. Gomes, J.-B. Mouret, and S. Ivaldi, "Learning robust task priorities and gains for control of redundant robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2626–2633, 2020.

[17] M. P. Deisenroth, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[18] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 328–347, 2020.

[19] A. Wilson, A. Fern, and P. Tadepalli, "Using trajectory data to improve bayesian optimization for reinforcement learning," *Journal of Machine Learning Research*, vol. 15, no. 8, pp. 253–282, 2014.

[20] R. Antonova, A. Rai, and C. G. Atkeson, "Sample efficient optimization for learning controllers for bipedal locomotion," in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2016.

[21] R. Antonova, A. Rai, and C. G. Atkeson, "Deep kernels for optimizing locomotion controllers," *CoRR*, 2017.

[22] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[23] 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 *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[24] K. Miettinen, *Nonlinear Multiobjective Optimization*. Springer US, 1998.

[25] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *CoRR*, 2009.

[26] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 5–23, Jun 2015.

[27] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv:1012.2599*, 2010.

[28] A. D. Prete, N. Mansard, O. E. Ramos, O. Stasse, and F. Nori, "Implementing torque control with high-ratio gear boxes and without joint-torque sensors," *Int. Journal of Humanoid Robotics*, vol. 13, no. 01, p. 1550044, Mar 2016.

[29] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE/SICE Int. Symposium on System Integration (SII)*, 2019.

[30] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. D. Prete, P. Soueres, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro, "TALOS: A new humanoid research platform targeted for industrial applications," in *IEEE-RAS Int. Conf. on Humanoid Robotics (Humanoids)*, 2017.

[31] J.-B. Mouret and S. Doncieux, "Sferes$_{v2}$: Evolvin' in the multi-core world," in *IEEE Congress on Evolutionary Computation*, 2010.

[32] D. Roetenberg, H. Luinge, and P. Slycke, "Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. xsens motion technologies bv," Tech. Rep., 2009.

[33] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.