2017 IEEE-RAS 17th International Conference on
Humanoid Robotics (Humanoids)
Birmingham, UK, November 15-17, 2017

Trial-and-Error Learning of Repulsors for Humanoid QP-based Whole-Body Control

Jonathan Spitz Karim Bouyarmane Serena Ivaldi Jean-Baptiste Mouret

Abstract— **Whole body controllers based on quadratic programming allow humanoid robots to achieve complex motions. However, they rely on the assumption that the model perfectly captures the dynamics of the robot and its environment, whereas even the most accurate models are never perfect. In this paper, we introduce a trial-and-error learning algorithm that allows whole-body controllers to operate in spite of inaccurate models, without needing to update these models. The main idea is to encourage the controller to perform the task differently after each trial by introducing repulsors in the quadratic program cost function. We demonstrate our algorithm on (1) a simple 2D case and (2) a simulated iCub robot for which the model used by the controller and the one used in simulation do not match.**

I. Introduction

Like humans, humanoid robots need to carefully coordinate dozens of degrees of freedom for even the most basic tasks, like standing upright or walking [1]. This challenge can be tackled in a principled way with Quadratic Programming-based Whole-Body Controllers (QB-based WBC) [2]: at each time-step, an optimizer minimizes a cost function that describes the task(s), under constraints that model the dynamics of the robot interacting with its environment [3]. This optimization is performed many times per second by formulating the problem as a quadratic program with linear constraints, which can be solved efficiently on modern computers.

The most fundamental assumption of QP-based WBC is that the model accurately captures the dynamics of both the robot and the environment. Unfortunately, no model is ever perfect and thus such controllers often fail when the real world and the model do not match, even when using accurate state estimators and even when assuming perfect state feedback from the robot (which allows for small perturbations to be absorbed to a certain extent). For instance, the robot's dynamics (contact forces, center of mass, friction cones, *etc.*) is rarely known with precision. Even with a good dynamics model extracted by CAD and refined by dynamics parameters identification [4], [5], there are elasticities, nonlinearities and coupled dynamics effects which are impossible to model and measure accurately on a complex platform like a humanoid, especially in presence of multiple contacts [6]. Some parts of the environment are also generally unknown, like the exact mass of objects or the friction of the floor, or might not be modeled accurately because of imperfect sensors. Ultimately, the robot may also be damaged [7], or some parts may be worn out. Overall, in practice, setting a QP-based WBC for humanoids almost always involves long hand-tuning sessions of the model and the cost function [8].

e-mail: `firstname.lastname@inria.fr`
`karim.bouyarmane@loria.fr`
All authors have the following affiliations:
- Inria, Villers-lès-Nancy, F-54600, France
- CNRS, Loria, UMR 7503, Vandœvre-lès-Nancy, F-54500, France
- Université de Lorraine, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
This work received funding from the Grand Est region (project "LocoLearn"), Inria (project "wbCub"), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (GA no. 637972, project "ResiBots"), and the European Commission (GA no. 731540, H2020 project "An.Dy").

The fact that a WBC can fail when its model is imperfect is not a problem *per se*: humans often fail when they have imperfect information or when their "internal model" is different (*e.g.*, when they move under perturbations [9] or after an injury). Humans, however, *learn from their mistakes*, *i.e.* they adapt their behavior until they find a way to achieve their objective. By contrast, a QP-based WBC with a fixed model and tasks structure will keep performing the same faulty behaviors.

Ideally, we would like to see humanoid robots that (1) attempt to achieve a whole body task with their WBC, (2) fail (*e.g.*, fall down), and (3) try again until they achieve the desired task. We would also like the learning process to succeed after only a few trials (less than a dozen) and a few minutes [10], [11], [12], in particular because of the limited energetic autonomy of robots. The main question here is: "how to incorporate new information from the real world into a QP-based WBC?"

Since a QP-based WBC assumes a perfect QP optimizer, only two elements can be updated in such a trial-and-error process: the cost function and the model (*i.e.*, the constraints). The most classic approach is to update the model according to the data acquired during each trial, *i.e.* perform a classic *model identification* [13]. Nevertheless, identifying the model of a full humanoid is far from being straightforward, as (1) identification can seldom be performed with only proprioceptive sensors [4], and (2) for a humanoid structure, it might require exciting the system in specific ways which may be unsafe for the robot [14]. More importantly, some effects cannot be captured by tuning the parameters of classic models. To take an extreme example, a humanoid walking through a flooded room would experience significant viscous forces, which are usually not taken into account in the classical model for the robot moving in an indoor environment.

Our main insight is that even if a model makes inaccurate predictions for some behaviors, this is not necessarily the case for all the behaviors [15], [16], [7], [17]. In the previous flooded room example, the viscous forces are likely to be negligible if the robot is moving slowly; therefore a learning process could discover that the WBC works fine when the robot moves slowly, without needing to update the model to take viscous forces into account. Similar situations are common when robots perform highly dynamic motions: while a perfect model might be required to move at high speed, a less accurate model might be enough for a lower speed motion. The redundancy of the robot also implies that many tasks can be achieved in different ways with equivalent costs. In case of damage, the redundancy of the robot plays a similar role: the behaviors that do not use the damaged parts (*e.g.*, a blocked joint) will be accurately predicted by the model, whereas those that rely on broken ones will have very different outcomes [15], [7]. In other words, we can use an imperfect model *if we know its limits*.

In the present paper, we introduce this idea in the QP-based WBC framework for whole-body motion, leading to a novel learning approach. When the robot performs a whole-body movement that, at some point, fails (for example, the robot falls), it should try to avoid repeating the same behavior. This "information" should be added to the QP problem so that the controller takes into account the "failures". The key challenge is that we cannot simply add penalties for failed states because the "bad decision" (*i.e.*, the control command and the state that caused at some point the failure) is very likely to have happened much before the moment when the robot actually "fails" or falls. In essence, this problem is a form of the classic "temporal credit assignment problem" in reinforcement learning [18], which is one of the most important problems of the field. For example, if a humanoid robot falls on the ground, then it reaches a failed state when it hits the ground; but penalizing the failed state would mean repulsing from the ground, which is something that we already know and is already identified as a bad state. Instead, at some point of its trajectory, the robot took a series of decisions that led to the failed state: these are the states that should be avoided. In classic reinforcement learning (e.g. Q-learning, see [18]), we would propagate the knowledge of the failed state to the previous state and start a new episode, but this means a prohibitive number of trials for a complex humanoid robot with many degrees of freedom (thousands in a complex space).

Our main concept is that we can update the cost function of the QP controller after each trial so that the robot tries to solve the task *in different ways* until it finds one that is satisfying. To do so, we use the state-space trajectory of failed trials to introduce *repulsors* along the trajectory that push the QP controller away from states that have already been visited, while still attempting to solve the task. Between each episode, a gradient-free optimization algorithm searches for the best repulsors so that the next trial is as different as possible while still "solving" the task. This new episodic, trial-and-error algorithm enables QP-based whole-body controllers to adapt in a few trials to unknown situations, like damage, and to imperfect models of the robot or its environment.

We show the effectiveness of our approach on a toy-example (a particle moving in 2D) and on a simulated iCub humanoid robot for a squatting task, in a situation where the robot model is wrong: the robot's feet are smaller in reality than in the QP model, which causes the QP controller to fail.

II. Background

A. Humanoid QP-based whole-body control

Whole-body control for humanoid robots is essentially formulated by prioritized multi-task controllers with strict task priorities [19] or multi-task controllers with soft task priorities (also called weights) [20], [21]. A now well-established classical formulation for the latter is the quadratic programming (QP) formulation, which solves, at every timestep of the control loop, the following QP problem:

$$\min_{\ddot{q},f,\tau} \sum_{k=1}^{m} w_k \left\| \ddot{g}_k - \ddot{g}_k^d \right\|^2 \tag{1}$$

$$\text{under constraints} \quad M\ddot{q} + N = J^T f + S\tau \tag{2}$$

$$J\ddot{q} + \dot{J}\dot{q} = 0 \tag{3}$$

$$f \in K \tag{4}$$

$$q_{\min} \leq q \leq q_{\max} \tag{5}$$

$$\dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max} \tag{6}$$

$$\tau_{\min} \leq \tau \leq \tau_{\max}\,, \tag{7}$$

where q is the configuration of the robot (including the 6D position and orientation of the floating-base), $\dot{q}$ and $\ddot{q}$ are (as an abuse of notation) the configuration-space velocity and acceleration (thus respectively including the angular velocity and angular acceleration of the base, which are not the time derivatives of the base orientation representation), M is the inertia matrix, N the gravity and Coriolis term, J the stacked Jacobian matrices at the contact points, f the vector composed of all the point contact forces, τ the actuator torques, S a selection matrix that maps the dimension of the actuated joints $n-6$ to the dimension of the configuration space n, K the Cartesian product of the linearized Coulomb friction cones at the contact points. Equations (5) and (6) are converted into linear constraints on $\ddot{q}$ by using finite differentiation and velocity dampers [22].

A task g_k, with its relative weight w_k, is any mapping $\mathbb{R}^n \to \mathbb{R}^{n_k}$, where n_k is the dimension of the task. A task is characterized by its Jacobian matrix $J_k = \frac{\partial g_k}{\partial q}$ and a desired acceleration behaviour $\ddot{g}^d$. A classical behaviour is the *attractor behaviour*, which is used to bring the task to a desired set-point value g_k^{ref}. The attractor behaviour writes:

$$\ddot{g}_k^d \triangleq -\alpha_k (g_k - g_k^{\text{ref}}) - 2\sqrt{\alpha_k}\dot{g}_k\,. \tag{8}$$

It models a critically-damped mass-spring-damper system with unit apparent mass, parameterizable stiffness α_k, and critical damping $2\sqrt{\alpha_k}$. This is the most used task in QP controllers. We use it in the examples below to make the humanoid robot perform a squatting motion. The attractor task in this example is on the center-of-mass (CoM) c with reference setpoint that is lowered from the standing up posture CoM $c^{\text{ref},1}$ to the crouching posture CoM $c^{\text{ref},2}$.

As opposed to using nonlinear constrained robotics optimisation frameworks such as [23], a QP can be solved in few milliseconds and thus allows for online control.

B. Learning with humanoid robots

A large part of the work about learning motion controllers for humanoid robots has focused on the optimization of neural networks [24], [25] or Central Pattern Generators (CPG) [24], [26] in simulation, with the hope of transferring the optimized controller to the physical robot; some others rely on more classic reinforcement learning approaches (e.g. Q-learning) and learn discrete policies (e.g., [27]). However, all these approaches assume that the simulation perfectly captures the dynamics of the robot. The few experiments performed with real robots involve learning very few parameters (typically from 2 to 4) for simple walking patterns [28], [29], because only a few dozen of controller evaluations can be reasonably performed with a robot.

While interesting from an artificial intelligence perspective, these learning approaches are far from being competitive with QP-based WBC: they can only generate simple motion for a single task (e.g. walking forward) and are often open-loop. In addition, they are not compatible with QP-based WBC, which prevents the two communities to exchange ideas and progress together.

A handful of papers combine QP-based WBC with some form of learning (e.g. [30]). Modugno *et al.* [31] exploited a black-box optimizer (CMA-ES [32]) to learn the temporal profiles of the task weights of a QP-based controller, using a simulated robot. In [8] they used constrained variants of CMA-ES to learn safe task weights that guarantee that the controller never violates constraints. However, the optimization was still done offline and not on the real robot. Mukovskiy *et al.* [33] learned movement primitives from human motion captures, and combined them with model predictive control and planning to generate whole body motions on the HRP-2 robot. Clever *et al.* [34] used motion generated in simulation with a QP-based WBC to learn motion primitives, which can then be employed instead of the QP-based WBC while having a lower computational cost. Overall, these papers use learning in simulation only and aim at making whole body control more reliable and less computationally demanding. Contrary to the present paper, they do not aim at incorporating data from failed executions into the controller itself. Although not tested on a physical robot (to our knowledge), the approach proposed by Lober *et al.* [35] is probably the closest to the present work. In this work, the authors used Bayesian Optimization (BO) [7], which is a model-based black-box optimization algorithm, to move an intermediate waypoint for a QP-based WBC. However, as noted by the author, BO does not scale besides a few parameters (about 5 to 10 parameters to optimize). The same observation was done by Antonova *et al.* [36] that used BO and CMA-ES to optimize the 16 parameters of a walking controller for a simulated planar robot.

Finally, in a study related to the present paper, but not directly related to learning, Gori *et al.* used force fields as attractors and repulsors to model targets and obstacles inside a reaching/tracking controller for the upper-body of iCub [37]. The authors, however, only applied their approach to the upper-body of the robot on a fixed base, and relied on real-time perception of visual targets and obstacles. In other words, they did not attempt to *learn*, that is, to improve performance after several trials; instead, they only adapted the current controller to the environment.

C. Transferability in robot learning

It is well documented that optimizers overfit simulators most of the time, which leads to behaviors that are high-performing in simulation, but low-performing in reality. This issue is commonly called the "reality gap" [38], [16], [39].

Many ideas have been explored to cross this "reality gap", especially in evolutionary robotics. The proposed solutions range from adding noise to the simulation [38], [40] to automatically improving simulators [41], [11]. One of the most successful approaches is the "transferability approach" [16], [15], [39]: instead of attempting to correct the simulator, the transferability approach hypothesizes that the simulator is accurate for *some* behaviors and not others; it is therefore possible to (1) learn the limits of the simulation, which is a supervised learning problem, and (2) encourage the learning process to select behaviors that are within these limits.

This idea was recently used to learn new walking policies for a damaged 6-legged robot in less than 2 minutes / a dozen trials [7]. In these experiments, the robot can find a working policy because some behaviors from the simulated, intact robot perform the same on the damaged, real robot (typically because they do not rely on the damaged part). Put differently, the algorithm finds the subset of high-performing solutions that work similarly in the simulated, intact robot and the physical, damaged robot. However, this algorithm does not fit the QP-based WBC framework, since it is designed to learn parametrized policies (e.g. CPGs or neural networks). In the present work, we exploit the same concept of transferability, but we formulate it in a way that leverages the benefits of modern WBC.

III. REPULSOR LEARNING

A. Repulsor behavior with the QP controller

When the robot executes a QP-controlled motion that leads to a failure (e.g. the robot falling down, the QP failing to find a solution, *etc.*), we want to instruct the controller to *avoid* that motion. Let $t \mapsto q^{\text{fail}}(t)$ be the trajectory in the configuration space, corresponding to such an ultimately failing motion. The trajectory is discretized in a number n_r of *repulsor* configurations $q_1^{\text{fail}}, \ldots, q_{n_r}^{\text{fail}}$, at times $t_1, \ldots, t_{n_r}$ distributed uniformly along the total time of the motion.

In order to write a QP-compatible repulsor task from a repulsor configuration q_j^{fail}, we take inspiration from the behavior of the Coulomb's inverse-square law between electrically charged point particles. Considering the current configuration q and the the repulsor configuration q_j^{fail} as two positive electric charges, the desired behavior writes:

$$\min_{\ddot{q}} \left\| \ddot{q} - \frac{\Delta}{\|\Delta\|^3} \right\|^2 , \quad \Delta \triangleq \beta \left(q - q_j^{\text{fail}} \right), \tag{9}$$

where β is a diagonal matrix with non-negative, and at least one positive, elements. Changing the elements in β affects the repulsion strength along each coordinate, which can also be set to zero. Let us denote the desired acceleration of the repulsor behavior from q_j^{fail} as:

$$\ddot{q}^{\text{rep}}(q_i^{\text{fail}}) \triangleq \frac{\Delta}{\|\Delta\|^3} . \tag{10}$$

Adding the set of repulsors from the failed motion to the QP (1) with respective weights $(w_1^{\text{rep}}, \dots, w_{n_r}^{\text{rep}})$, the latter stays definite positive and becomes:

$$\min_{\ddot{q},f,\tau} \sum_{k=1}^{m} w_k \left\| \ddot{g}_k - \ddot{g}_k^d \right\|^2 + \sum_{j=1}^{n_r} w_j^{\text{rep}} \left\| \ddot{q} - \ddot{q}^{\text{rep}}(q_j^{\text{fail}}) \right\|^2 \tag{11}$$

under constraints (2) to (7).

As described in Section II-A, a classical QP task behavior is to attract the task to a desired set-point, using spring-damper dynamics (8). When there is a mismatch between the QP model and reality, this behaviour can cause the robot to fail. Our insight is that using the data from failed attempts to add repulsor tasks will cause the QP to explore away from previously visited states, while still trying to complete the task. The individual joint gains (β) determine how much each joint is repulsed by the repulsor, while the task weights (w_j^{rep}) determine the strength of each repulsor.

Fig. 1 illustrates this behaviour in a simple two-dimensional state-space where there is a mismatch between the QP model and the real world. In example A, the hatched area represents the mismatch between the real world robot and its simulated model (*i.e.*, the model used to compute the controls) which drives the state "downwards" preventing the QP to reach the task's goal. Adding repulsors on the failed trajectory causes the QP to explore away from the hatched area and complete the task (A.2). In the second case (B.1 and B.2), we reach a failed state when entering the hatched area. Adding the repulsors (red triangles) effectively changes the path followed by the QP in B.2, which approaches the goal from another configuration.

B. *Optimizing the repulsors*

The position of the added repulsors is determined by the trajectories previously explored by the QP-based WBC. However, the individual joint gains (β) and the weight of each repulsor task (w_j^{rep}) need to be tuned: if they are too high, the QP-based WBC will not solve the task (the repulsors will dominate the cost function of the quadratic program); if they are too small, then the next behavior is likely to be very close to the current failed ones, and therefore likely to fail as well. As a consequence, we need to optimize the gains and the weights so that the QP-based WBC solves the task as best as possible, while being as far as possible from the previous failed attempts.

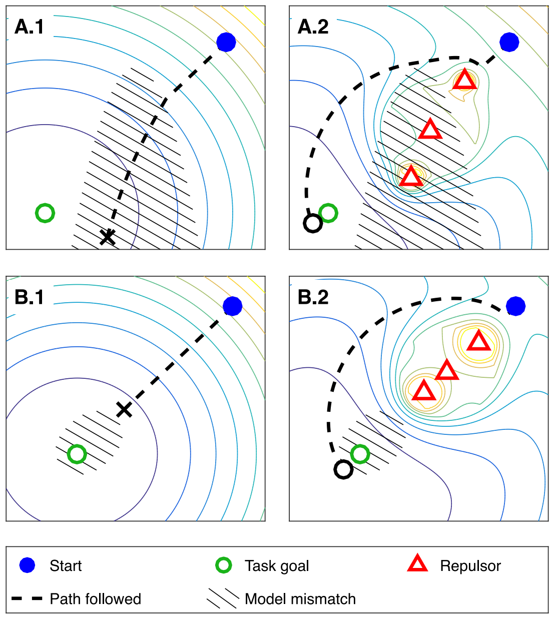


Fig. 1. Intuition of the repulsors approach, shown in a 2D state-space for simplicity. Each panel shows the path followed by the state from the start (blue, filled) to the task goal (green, hollow). Hatched areas show a mismatch between the QP model and the real world. Repulsors are shown with red triangles.

These two objectives can be evaluated for a vector of individual joint gains diag(β) and a set of $i \cdot n_r$ repulsor weights $W = [w_1^{\text{rep}}, \dots, w_{i\times n_r}^{\text{rep}}]$ by running the QP controller (Eq. 11) for n_T time-steps *without interacting with the physical robot*, that is, by simulating the robot with the same model as the QP controller, and computing the two following cost functions: (1) the average distance of the QP-generated trajectory from each repulsor (c_E) and (2) the tracking error of the QP task (c_T). More formally, we compute:

$$c_E(\beta, W) \triangleq \frac{1}{i \cdot n_r} \int_0^T \sum_{j=1}^{i \cdot n_r} \exp\left(- \left\| q(t) - q_j^{\text{fail}} \right\|^2 \right) dt \tag{12}$$

$$c_T(\beta, W) \triangleq \int_0^T \sum_{k=1}^{m} w_k \left\| g_k(t) - g_k^d \right\|^2 dt , \tag{13}$$

where i is the current episode number and T is the duration of each episode.

We normalize c_E and c_T using a Monte Carlo estimate of the bounds for each cost. Specifically, we run the QP controller K times (e.g., $K = 100$) with random values for β and W; estimate c_E and c_T for each run; and finally normalize them using the 5^{th} and 95^{th} percentiles (to avoid outliers):

$$C_E(\beta, W) \triangleq \frac{c_E(\beta, W) - I_{5\%}(c_E)}{I_{95\%}(c_E) - I_{5\%}(c_E)} \tag{14}$$

$$C_T(\beta, W) \triangleq \frac{c_T(\beta, W) - I_{5\%}(c_T)}{I_{95\%}(c_T) - I_{5\%}(c_T)} \tag{15}$$

where $I_{5\%}(x)$ denotes the 5^{th} percentile of x and $I_{95\%}(x)$ the 95^{th} percentile.

Lastly, we combine C_E and C_T in a single cost function:

$$C(\beta, W) \triangleq \lambda \cdot C_E(\beta, W) + (1 - \lambda) \cdot C_T(\beta, W) + P(t) \tag{16}$$

where λ is a user-defined parameter that represents the trade-off between the two cost functions, *i.e.* between exploration and exploitation. A penalty $P(t)$ is added to the cost if a failed state is reached before the end of the allotted episode time:

$$P(t) = 2\frac{t}{T_{\text{allotted}}} \tag{17}$$

$C(\beta, W)$ is not differentiable since it involves solving a QP program at every time-step. We therefore consider it as a black-box function that we optimize with CMA-ES[1] [32], a state-of-the-art global optimizer for non-linear, stochastic, black-box optimization.

This optimization is performed by running simulations using the QP model and different sets of repulsor parameters. This process typically requires hundreds of calls to the cost function, which makes the overall optimization computationally demanding (at least several minutes for a humanoid using a modern computer). Nevertheless, our aim is primarily to reduce the *interaction time*, that is, the time spent trying controllers on the real robot, because we assume that computers will be faster and faster in the future. In addition, CMA-ES can take full advantage of multi-core computers and clusters to speed up the computation. It is also important to highlight that the optimization becomes harder after each episode, since we increase the number of parameters to optimize after each episode on the physical robot. Our approach is therefore not likely to work with more than a dozen of episodes; it still fits our use case well because we aim at performing at most a dozen of learning episode with the physical robot.

C. Episodic learning

The learning algorithm searches for high-performing, alternative control strategies until it has exhausted its budget of episodes (or until a satisfying solution is found). Algorithm 1 details the full learning loop, which is based on three main steps:

1) attempt to complete the task using the QP controller in the real world (with the physical robot); record the trajectory and the tracking cost $c_T(\beta, W)$ (line 6);
2) add new repulsors $\mathscr{R}$ using the state-space trajectory $q^{(i)}$ of the last episode (line 11);
3) optimize the parameters of the repulsors (β, W) with CMA-ES and the QP controller, using the model of the controller as a simulator (line 12).

After each episode, the robot is reset to its initial position. A success threshold for the tracking cost is not specified to keep a more general approach. In this case, each trajectory is considered as sub-optimal and is therefore avoided in future episodes. Thus, the output of the algorithm is not necessarily the last set of repulsors, joint gains, and repulsor weights, but the one that correspond to the best tracking cost in the real world.

[1]We use the implementation of the libcmaes library: `https://github.com/beniz/libcmaes`

Algorithm 1 Repulsor learning

```
 1: procedure LEARN
 2:     $\mathscr{R} \leftarrow \emptyset$                              ▷ set of repulsors
 3:     $C^* \leftarrow \infty$                              ▷ best cost so far
 4:     $\beta^*, W^* \leftarrow 0, 0$                        ▷ best parameters
 5:     for $i$ from 1 to $N$ do                  ▷ for each episode
 6:         $q^{(i)} \leftarrow$ Run episode in real world using QP
        controller with $\mathscr{R}(\beta, W)$ and compute $c_T(\beta, W)$ (Eq. 13)
 7:         if $c_T < c_T^*$ then      ▷ update the best parameters
 8:             $\mathscr{R}^* \leftarrow \mathscr{R}$
 9:             $\beta^*, W^* \leftarrow \beta, W$
10:             $c_T \leftarrow c_T^*$
11:         for $j$ from 1 to $n_r$ do           ▷ add new repulsors
12:             $\mathscr{R} \leftarrow \mathscr{R} \cup q^{(i)}(t_j)$, $t_j = \frac{j-1}{n_r-1}T$
13:         $\beta, W \leftarrow \text{argmin}(\text{SIMEPISODE}(\mathscr{R}))$    ▷ CMA-ES
14:     return $\mathscr{R}^*, \beta^*, W^*$
15: procedure SIMEPISODE($\mathscr{R}$)
16:     Run episode using QP model, controlled by QP with
    $\mathscr{R}(\beta, W)$ (Eq. 11)
17:     Calculate exploration cost, $C_E$, using (Eq. 12, 14)
18:     Calculate task cost, $C_T$, using (Eq. 13, 15)
19:     return $C(\beta, W)$ (Eq. 16)
```

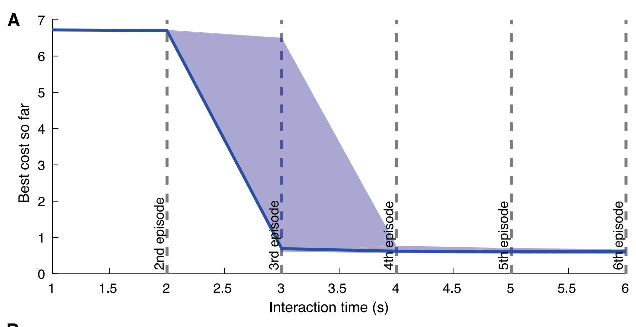
IV. Experimental Results

A. 2D particle

Our first objective is to validate the concept of repulsor learning on a simple and visual example (Fig. 2). To do so, we implemented our algorithm for a simple two-dimensional particle of mass 1 kg that has to start from state $(1, 0)$ and reach state $(-1, 0)$. At each time-step, the QP controller chooses the magnitude of a 2D force, which is applied on the particle with the standard equation of dynamics. To introduce a mismatch between the QP model and the "real world", we placed a circular obstacle of radius 0.3 around $(0, 0)$ which is unknown to the QP controller. Hitting the obstacle puts the particle in a "failed state", which halts the controller. In higher-dimensional spate spaces, the equivalent of this obstacle would be a zone in which the robot fails, for instance, a zone of the state-space in which a humanoid robot fails to maintain its balance. After each episode, we add 6 repulsors, resulting in 8 to 32 parameters to optimize[2], depending on the episode number. Since the optimization procedure is stochastic, we replicate each experiment 20 times to gather statistics. To optimize the cost, CMA-ES is given a budget of 500 calls to the cost function.

The results show that three episodes are enough to reach the target state (Fig. 2A). In a typical run (Fig. 2B), the particle first hits the obstacle (for which there is a mismatch between the model and the "reality"), which is expected since the controller has no knowledge of the obstacle. Repulsors are then added, but the second episode is often too close to the obstacle. At the third episode, the particle usually reaches

[2]The particle has a 2-dimensional state. The number of parameters is $2 + 6 \times i$, where 6 is the number of repulsors and i the learning episode.



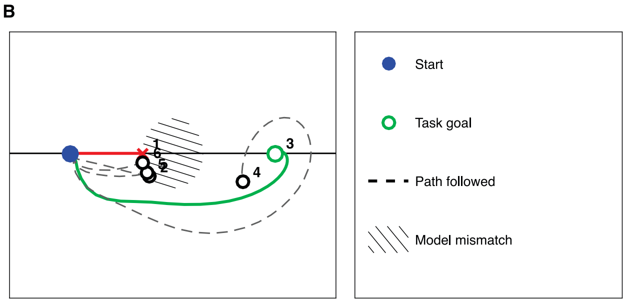


Fig. 2. Tracking cost for the particle task (20 replicates). **A**: Median of the best tracking cost since the beginning of the learning process and 25th-75th percentiles. The task is solved after only 3 to 4 seconds of interaction time (3 to 4 episodes). **B**: Behavior of a typical controller. The first two episodes (trajectories 1 and 2) hit the obstacle; the third tested trajectory solves the task and obtains the best tracking cost; the fourth trajectory is very different from the third trajectory, but it obtains a higher cost. The selected controller is the third one.

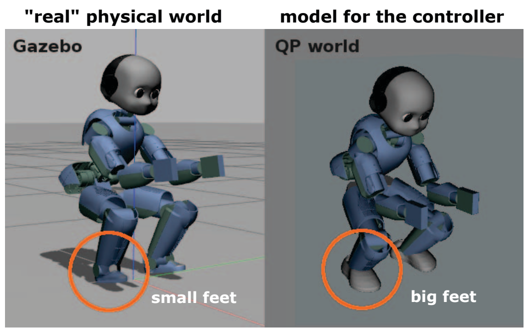


Fig. 3. The mismatch between the “real world” iCub (left, small feet) and its internal model in the QP controller (right, large feet) causes the iCub robot to fail in the real world, even though the QP-based WBC generates a feasible joints trajectory. The “real world” here is in Gazebo.

the target state. After this episode, the algorithm explores more complex trajectories, but the resulting trajectories often have a high tracking error because they have to be different from the “good” trajectory that was found at the third episode.

B. *iCub squatting with a mismatch in feet geometry*

We then evaluate our approach with a QP-based WBC for the iCub robot, with a mismatch between the internal model of the QP controller and the model simulated in Gazebo[3]. iCub is a 53-DOF full humanoid robot, with 7-DOF arms, 3-DOF torso and 6-DOF legs. It is fully open-

[3]In this case, the simulated robot in Gazebo is the “real world” robot.

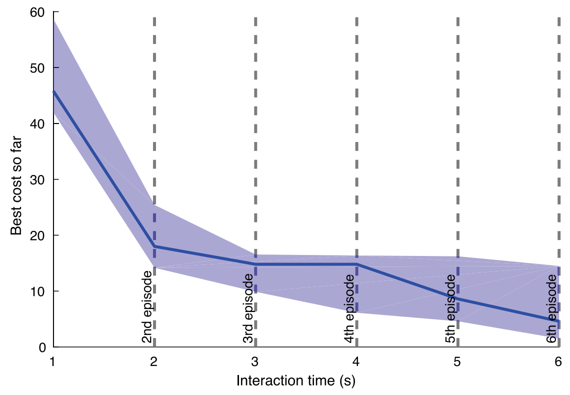


Fig. 4. Tracking cost for the iCub squatting task (13 replicates). Median of the best tracking cost since the beginning of the learning process and 25th-75th percentiles. The task performance increases (lower cost) on average as more trials are performed, with the major improvement between the first and second trials.

source in both mechanics and software [42]. Its dynamics model, represented in a URDF format, is used by a software abstraction layer to control the simulated robot in Gazebo as if it was the real one [43]. Wrenches, contact forces and center of mass are computed for both the real and simulated robot by open-source libraries for dynamics estimation [44]. To optimize the cost, CMA-ES is given a budget of 500 calls to the cost function.

We created a QP-based WBC for iCub using the MC-RTC framework [3]. The controller is composed of a position task for the robot’s center of mass (CoM), the left hand and the right hand. Each position task switches its set-point position every t_{period} seconds, taking iCub from a standing to a crouching configuration and back. Specifically, we move the CoM from $y = 0.45$m to $y = 0.3$m. When both the QP controller and Gazebo, *i.e.*, the “real world”, have the exact same model, iCub is able to perform the squatting task with the “classic” QP controller with three tasks (and no repulsors). However, we introduce a mismatch to the QP model that causes iCub to fail when performing the task. Specifically, we provide iCub with a different (smaller) pair of feet, as shown in Figure 3, which is a common occurrence in this platform as different variations of it are used in labs around the world. The smaller feet are 15 cm long and 5 cm wide while the bigger feet which are 19 cm long and 6.5 cm wide. The model in the QP controller no longer reflects the real world robot: as expected, the “classic” QP controller with the three tasks fails, and the robot falls.

Therefore, we start learning the repulsors as described in Section III. The controlled robot has 53DOF, which would result in a 53-dimensional repulsor if all the joints would be considered (more if the full state would be considered, including velocities and accelerations). However, most of the joints of iCub are not “critical” for balancing (e.g., 9 DOF in each hand). For this reason, and for keeping the number of parameters to optimize as low as possible, we designed a repulsor state of 8-DOF, with the following joints: [“torso_pitch”, “torso_roll”, “torso_yaw”, “neck_pitch”, "l_hip_pitch", "r_hip_pitch", "l_knee", "r_knee"].

We run the learning algorithm for 6 episodes, adding 6 new repulsors after each episode and optimizing their parameters before making a new attempt at fulfilling the task with the "real robot". The number of parameters to optimize is therefore $\dim(\beta) + \dim(W) \times i = 8 + 6 \times i$, ranging from 14 ($i = 1$, first episode) to 38 ($i = 5$, last episode).

There is a clear improvement in the task performance when using repulsors, as seen in Fig. 4. As more trials are performed and new repulsors are added, the performance increases further. Although the robot is not able to perform the same squatting motion as expected by the QP-based WBC (because the QP controller is using a wrong model), it is still able to perform a comparable squatting motion without falling down (see the Supplementary Video). The CoM trajectories from one learning process are shown in Fig. 5. Here we can see that the original trajectory, *i.e.* without repulsors (shown in blue), is not able to make the robot stand up after performing the squat (the robot falls). In three of the following trials iCub performed a smaller squat (see the Supplementary video) and was able to go back to a standing position.

V. Conclusion and discussion

Learning repulsors is a novel and promising approach to mix QP-based WBC with trial-and-error learning. It enables applying QP controller to real world robots, despite the inevitable model errors, and without requiring to identify or improve the robot model used by the QP controller. In the two experimental setups of this paper, only 3 to 5 episodes are required to find a controller that works in spite of an important mismatch between the model used in the controller and the real world. This low number of episodes makes it easy to use our approach on physical humanoids robot and significantly lowers the risk of breaking them. In the iCub case, dealing with smaller feet is a clear example to demonstrate our approach, albeit a simple one to solve by measuring the feet and updating the model. We are exploring other demonstrations for our approach, and especially on the real iCub.

Nevertheless, the way we learn repulsors is only a first "proof of concept" of the idea and we believe the concept can be improved in several ways. First, the trade-off between solving the task and exploring needs to be well-tuned to obtain satisfying results. In future work, we will explore the use of stack-based formulations of the whole-body problem [19] (instead of the weight-based approach followed here), which would change the way we combine the repulsors with the main task.

A second direction of improvement is to accelerate (and potentially improve) the optimization of the repulsors by using a more data-efficient optimization algorithm. For instance, Bayesian optimization [7], [29] with an appropriate prior could require fewer calls to the cost function to find high-performing solutions. In addition, the proposed algorithm does not use all the available information. For instance, the score achieved in the real world is currently ignored by the algorithm, whereas it could be used to guide the search for the best repulsors.

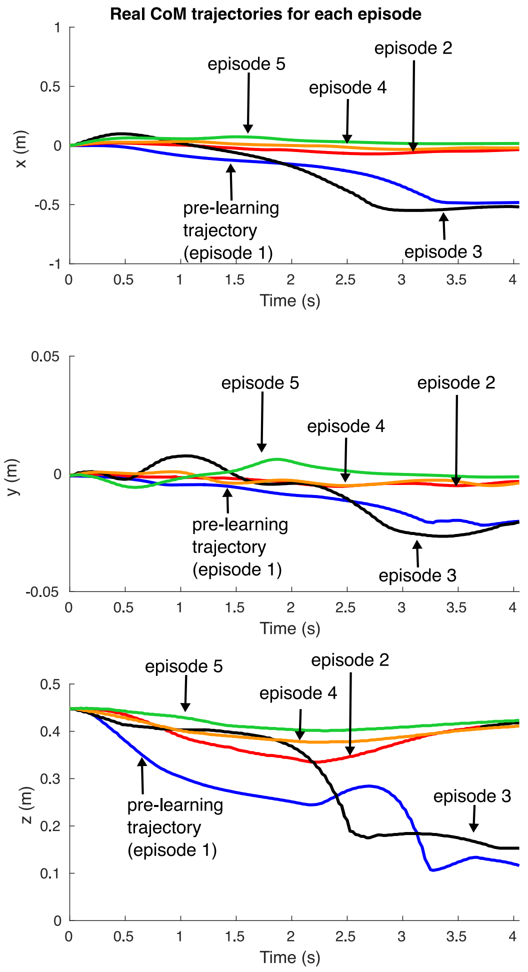


Fig. 5. Center of mass trajectories of the iCub robot for 5 episodes of a typical learning process in the "real world" (here the Gazebo simulation with small feet). The original trajectory, *i.e.* without repulsors, is shown in blue. Episodes 2, 4 and 5 led to satisfying behaviors (not falling, squatting movement). See the supplementary Video for a a visualization of the behaviors.

Overall, learning repulsors offers a new view of humanoid robot learning that bridges the gap between modern whole body control and reinforcement learning. We believe it opens many new research avenues to make humanoids robot that can both benefit from sophisticated control methods and adapt to unexpected situations.

VI. Acknowledgment

The authors would like to thank Brice Clement for the engineering support; the IDH team for their help with the QP controller (`https://github.com/jrl-umi3218/Tasks`); Gabriele Nava and Silvio Traversaro for their help with the iCub's URDF.

References

[1] S. Ivaldi, O. Sigaud, B. Berret, and F. Nori, "From humans to humanoids: the optimal control framework," *Paladyn*, vol. 3, no. 2, pp. 75–91, 2012.

[2] S. Ivaldi, J. Babič, M. Mistry, and R. Murphy, "Special issue on whole-body control of contacts and dynamics for humanoid robots," *Autonomous Robots*, vol. 40, no. 3, pp. 425–428, Mar 2016. [Online]. Available: http://dx.doi.org/10.1007/s10514-016-9545-5

[3] K. Bouyarmane and A. Kheddar, "On the dynamics modeling of free-floating-base articulated mechanisms and applications to humanoid whole-body dynamics and control," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012.

[4] S. Traversaro, A. Del Prete, R. Muradore, L. Natale, and F. Nori, "Inertial parameter identification including friction and motor dynamics," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. IEEE, 2013.

[5] S. Traversaro, A. Del Prete, S. Ivaldi, and F. Nori, "Inertial parameters identification and joint torques estimation with proximal force/torque sensing," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015.

[6] R. Calandra, S. Ivaldi, M. P. Deisenroth, E. Rueckert, and J. Peters, "Learning inverse dynamics models with contacts," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015.

[7] 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.

[8] V. Modugno, U. Chervet, G. Oriolo, and S. Ivaldi, "Learning soft task priorities for safe control of humanoid robots with constrained stochastic optimization," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016.

[9] E. Burdet, R. Osu, D. W. Franklin, T. E. Milner, and M. Kawato, "The central nervous system stabilizes unstable dynamics by learning optimal impedance," *Nature*, vol. 414, no. 6862, pp. 446–449, Nov. 2001.

[10] J.-B. Mouret, "Micro-data learning: The other end of the spectrum," *ERCIM News*, no. 107, p. 2, 2016.

[11] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," *Proc. of IEEE/RSJ IROS*, 2017.

[12] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 408–423, 2015.

[13] J. Hollerbach, W. Khalil, and M. Gautier, "Model identification," in *Springer Handbook of Robotics*. Springer, 2016, pp. 113–138.

[14] K. Yamane, "Practical kinematic and dynamic calibration methods for force-controlled humanoid robots," in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, 2011.

[15] S. Koos, A. Cully, and J.-B. Mouret, "Fast damage recovery in robotics with the t-resilience algorithm," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1700–1723, 2013.

[16] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2013.

[17] M. Oliveira, S. Doncieux, J.-B. Mouret, and C. P. Santos, "Optimization of humanoid walking controller: Crossing the reality gap," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. IEEE, 2013.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[19] A. D. Prete, F. Nori, G. Metta, and L. Natale, "Prioritized motion force control of constrained fully-actuated robots: a task space inverse dynamics," *Robotics and Autonomous Systems*, vol. 63, pp. 150 – 157, 2015.

[20] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[21] K. Bouyarmane and A. Kheddar, "Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011.

[22] J. Vaillant, K. Bouyarmane, and A. Kheddar, "Multi-character physical and behavioral interactions controller," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 6, pp. 1650–1662, 2017.

[23] T. Moulard, F. Lamiraux, K. Bouyarmane, and E. Yoshida, "Roboptim: an optimization framework for robotics," in *Robomec*, 2013.

[24] D. Hein, M. Hild, and R. Berger, "Evolution of biped walking using neural oscillators and physical simulation," in *Robot Soccer World Cup*. Springer, 2007.

[25] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems*, 2015.

[26] M. Oliveira, V. Matos, C. P. Santos, and L. Costa, "Multi-objective parameter cpg optimization for gait generation of a biped robot," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[27] H. Jeong and D. D. Lee, "Efficient learning of stand-up motion for humanoid robots with bilateral symmetry," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016.

[28] R. Tedrake, T. W. Zhang, and H. S. Seung, "Learning to walk in 20 minutes," in *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*. Yale University New Haven (CT), 2005.

[29] 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, 2016.

[30] K. Bouyarmane, J. Vaillant, N. Sugimoto, F. Keith, J.-i. Furukawa, and J. Morimoto, "Brain-machine interfacing control of whole-body humanoid motion," *Frontiers in systems neuroscience*, vol. 8, 2014.

[31] V. Modugno, G. Neumann, E. Rueckert, G. Oriolo, J. Peters, and S. Ivaldi, "Learning soft task priorities for control of redundant robots," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016.

[32] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation*, pp. 75–102, 2006.

[33] A. Mukovskiy, C. Vassallo, M. Naveau, O. Stasse, P. Soueres, and M. A. Giese, "Adaptive synthesis of dynamically feasible full-body movements for the humanoid robot HRP-2 by flexible combination of learned dynamic movement primitives," *Robotics and Autonomous Systems*, vol. 91, pp. 270–283, 2017.

[34] D. Clever, M. Harant, K. Mombaur, M. Naveau, O. Stasse, and D. Endres, "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 Robotics and Automation Letters*, vol. 2, no. 2, pp. 977–984, 2017.

[35] R. Lober, V. Padois, and O. Sigaud, "Efficient reinforcement learning for humanoid whole-body control," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016.

[36] R. Antonova, A. Rai, and C. G. Atkeson, "Sample efficient optimization for learning controllers for bipedal locomotion," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016.

[37] I. Gori, U. Pattacini, F. Nori, G. Metta, and G. Sandini, "Dforc: a real-time method for reaching, tracking and obstacle avoidance in humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots*, 2012.

[38] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," *Advances in artificial life*, pp. 704–720, 1995.

[39] J.-B. Mouret, S. Koos, and S. Doncieux, "Crossing the reality gap: a short introduction to the transferability approach," *arXiv preprint arXiv:1307.1870*, 2013.

[40] A. Del Prete and N. Mansard, "Robustness to joint-torque-tracking errors in task-space inverse dynamics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1091–1105, 2016.

[41] R. Moeckel, Y. N. Perov, A. T. Nguyen, M. Vespignani, S. Bonardi, S. Pouya, A. Sproewitz, J. van den Kieboom, F. Wilhelm, and A. J. Ijspeert, "Gait optimization for roombots modular robotsmatching simulation and reality," in *Proc. of IEEE/RSJ IROS*. IEEE, 2013.

[42] L. Natale, F. Nori, G. Metta, M. Fumagalli, S. Ivaldi, U. Pattacini, M. Randazzo, A. Schmitz, and G. Sandini, *The iCub Platform: A Tool for Studying Intrinsically Motivated Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 433–458.

[43] F. Romano, S. Traversaro, D. Pucci, J. Eljaik, A. Del Prete, and F. Nori, "A whole-body software abstraction layer for control design of free-floating mechanical systems," in *Robotic Computing (IRC), IEEE International Conference on*. IEEE, 2017.

[44] M. Fumagalli, S. Ivaldi, M. Randazzo, L. Natale, G. Metta, G. Sandini, and F. Nori, "Force feedback exploiting tactile and proximal force/torque sensing," *Autonomous Robots*, vol. 33, no. 4, pp. 381–398, 2012.