Learning soft task priorities for safe control of humanoid robots with constrained stochastic optimization

Valerio Modugno[1,2], Ugo Chervet[2], Giuseppe Oriolo[1], Serena Ivaldi[2]

***Abstract*— Multi-task prioritized controllers are able to generate complex robot behaviors that concurrently satisfy several tasks and constraints. To perform, they often require a human expert to define the evolution of the task priorities in time. In a previous paper [1] we proposed a framework to automatically learn the task priorities using a stochastic optimization algorithm (CMA-ES), maximizing the robot performance for a certain behavior. Here, we learn the task priorities that maximize the robot performance, ensuring that the optimized priorities lead to safe behaviors that never violate any of the robot and problem constraints. We compare three constrained variants of CMA-ES on several benchmarks, among which two are new robotics benchmarks of our design using the KUKA LWR. We retain (1+1)-CMA-ES with covariance constrained adaptation [2] as the best candidate to solve our problems, and we show its effectiveness on two whole-body experiments with the iCub humanoid robot.**

I. Introduction

Fulfilling multiple operational tasks to achieve a complex behavior while satisfying constraints is one of the challenges of whole-body control of redundant manipulators and humanoid robots. For example, let us consider the humanoid iCub (Fig.1) that must fulfil a "global task" *by reaching its hands towards two goal positions behind a wall while avoiding collisions*. The global task can be decomposed as a combination of simpler elementary tasks (for example: control the end-effector, control the pose of a particular link, *etc.*) and constraints that guarantee a condition of feasibility over the generated motions (for example: torque and joint limits, collisions, external forces *etc.*).

More generally, elementary tasks can include tracking desired trajectories, regulating contact forces, controlling the center of mass for balancing *etc*. Constraints range from mechanical limitations (*e.g.*, joint and torque limits) to safety specifications (*e.g.*, collision avoidance, limiting the exchange of mechanical forces with the environment) and balance keeping for floating base platforms.

In the literature, this constrained control problem is usually solved with prioritized controllers, where a set of operational tasks are organized according to strict priorities in a hierarchy or "stack" [3], [4], or combined with weighting functions, also known as soft task priorities [5], [6]. Constraints are either formulated as high-priority tasks or taken into account by quadratic programming solvers. The task priorities and their evolution in time are usually defined *a priori* and frequently manually tuned by experts.

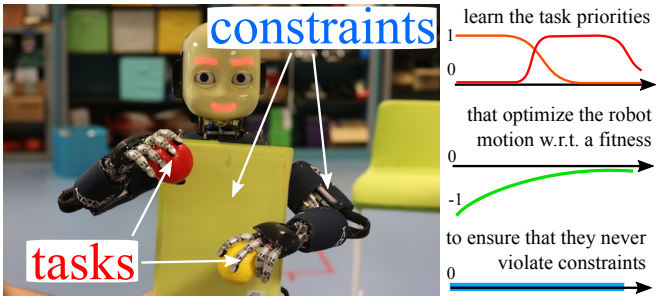

Fig. 1. The humanoid robot iCub performing a bimanual task with several tasks and constraints. In this paper we optimize the task priorities guaranteeing that the global robot behavior is safe: it never violates any of the constraints.

A new line of research is now focused on the automatic optimization of task priorities [1], [7], [8], [9]. Most of these approaches are based on an iterative policy learning technique that needs many repetitions (rollouts) of the same experiment to find a viable solution. These frameworks poorly address the problem of constraints satisfaction when optimizing the task priorities. For example, in [7], torques are saturated for safety, and joint and velocity limits are introduced as tasks. However, satisfaction of constraints formulated as tasks cannot be ensured, especially in the case of soft tasks prioritization. In [8] the balance constraint is added as an objective to the fitness function, but this is a relaxation of the constraint that does not ensure its satisfaction either. In [1] we used the Covariance Matrix Adaptation-Evolutionary Strategy (CMA-ES) [10], a derivative-free stochastic optimization method that solves non-linear, non-differentiable optimization problems, with death penalties to enforce constraint satisfaction on the solutions. This choice was not efficient in terms of searching for the optimum solution, since the exploration could easily get stuck in a constrained region where the fitness landscape was turned into a plateau. Furthermore, many solutions had to be dropped because of constraints violation.

Ensuring that the optimization process yields a safe solution — where safety means not violating any constraints — becomes mandatory if we want to successfully apply these solutions to a real robot [11].

To approach the safety issue, in this paper we investigate *constrained* stochastic optimization algorithms, and we focus on three variants of CMA-ES: one with *vanilla* constraints,

*This paper was supported by the EU FP7 project CoDyCo (n.600716) and by the EU H2020 project COMANOID (n.645097).

[1] Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Universita di Roma, via Ariosto 25, 00185 Roma, Italy. modugno@diag.uniroma1.it

[2] Inria, Villers-lès-Nancy, F-54600, France; CNRS, Loria, UMR n.7503 and Université de Lorraine. serena.ivaldi@inria.fr

The authors would like to thank the anonymous reviewers.

one with adaptive constraints [12] and the (1+1)-CMA-ES with covariance constrained adaptation [2]. We compare these methods with a baseline constrained optimization algorithm, (the *fmincon* function in Matlab). To compare the algorithms, we explicitly look for methods that can find good solutions while ensuring zero constraint violations within a reasonable computation time.

There exist standard benchmarks for constrained optimization, consisting in analytic problems with several variables and constraints and known optimal solutions. For example Arnold & Hansen [2] tested (1+1)-CMA-ES on seven different problems with a number of variables ranging from 2 to 10, and a number of constraints between 1 to 9. However, in robotics the number of constraints usually grows with the number of degrees of freedom (DOF) of the robot: for example, with a 7-DOF robot, the joint position range (7×2) and the torque limits (7×2) already introduce 28 constraints. In humanoids and highly articulated systems, the number of DOF is higher (*e.g.*, 32 DOF for the iCub) and so is the number of constraints. Furthermore, the number of tasks increases with the complexity of the action, especially for bimanual or whole-body movements. It is therefore necessary to design new benchmarks tailored for robotics applications to make a pondered decision about the algorithm that is most suited to solve our problem while ensuring that the constraints are never violated.

The contribution of this paper is twofold: first, we compare the performance of three constrained variants of CMA-ES with *fmincon* on analytic and robotic benchmarks, the latter (RB1,RB2) being new and designed *ad hoc*; second, we extend the framework for learning task priorities, which we proposed in [1], to ensure that the optimized priorities lead to safe behaviours that never violate the constraints. We show the effectiveness of our approach by generating optimized and safe (zero constraints violations) whole-body movements on the humanoid robot iCub.

The paper is organized as follows: Section II outlines the framework for learning task priorities for controlling redundant robots; Section III describes the constrained optimization algorithms retained for the study; Section IV and V illustrate the benchmarks comparison and the experiments with the iCub humanoid robot respectively.

II. Multitask controller with learnt priorities

Our method aims at automatically learning the task priorities (or task weight functions) to maximize the robot performance ensuring that the optimized priorities lead to behaviours that always satisfy the constraints. The global robot movement is evaluated by a fitness function ϕ that is used as a measure of the ability of the robot to fulfil its mission without violating the constraints. Our proposed method outlined in Fig. 2 extends the framework that was introduced in [1]. In this section we recall the multi-tasks controller and the structure of the parametrized task weight functions α_i, while the optimization procedure is described in Section III, where we analyze some recent extensions of the basic CMA-ES method that deal with constraints.

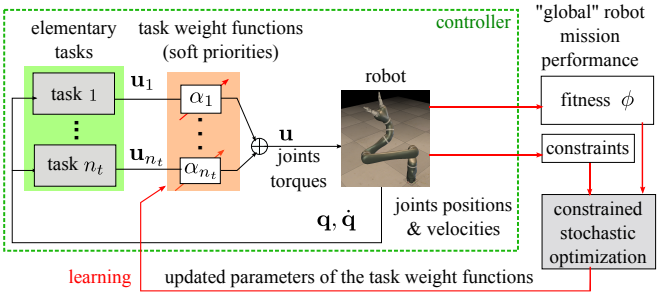

Fig. 2. Overview of the proposed method. The controller consists of a weighted combination of elementary tasks, where the weight functions represent the soft task priorities. An outer learning loop enables the optimization of the task weight parameters, taking into account the constraint violations in an explicit way.

A. Controller for a single elementary task

Here, we briefly describe the torque controller for the i-th elementary task, which is presented in more detail in [1]. Following our previous work, we use a regularized closed-form solution of a controller derived from the Unified Framework (UF) [13]. Let us consider the rigid-body dynamics of a robot with n DOF, *i.e.*:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}+\mathbf{f}(\mathbf{q},\dot{\mathbf{q}})=\mathbf{u}_i(\mathbf{q},\dot{\mathbf{q}})$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}\in\mathbb{R}^n$ are, respectively, the joints positions, velocities, and accelerations; $\mathbf{M}(\mathbf{q})\in\mathbb{R}^{n\times n}$ is the generalized inertia matrix, $\mathbf{f}(\mathbf{q},\dot{\mathbf{q}})\in\mathbb{R}^n$ accounts for Coriolis, centrifugal and gravitational forces; and $\mathbf{u}_i(\mathbf{q},\dot{\mathbf{q}})\in\mathbb{R}^n$ is the vector of the commanded torques of the i-th task. Following the UF formulation, the general torque controller is $\mathbf{u}_i=\mathbf{N}_i^{-\frac{1}{2}}(\mathbf{A}_i\mathbf{M}^{-1}\mathbf{N}_i^{-\frac{1}{2}})^{\dagger}(\mathbf{b}_i+\mathbf{A}_i\mathbf{M}^{-1}\mathbf{f})$, where the matrix $\mathbf{A}_i(\mathbf{q},\dot{\mathbf{q}},t)\in\mathbb{R}^{m\times n}$ and the vector $\mathbf{b}_i(\mathbf{q},\dot{\mathbf{q}},t)\in\mathbb{R}^{m\times 1}$ incorporate the information about the m-dimensional task; $\mathbf{N}_i$ is a weighting matrix that can be changed to achieve different control strategies; $(\cdot)^{\dagger}$ is the Moore-Penrose pseudoinverse; and the upper script in $\mathbf{N}_i^{-1/2}$ denotes the inverse of the matrix square root. Controllers derived from UF are sensitive to kinematic singularities, due to the matrix inversion [14]. To overcome this problem, we reformulate the UF controller in a *regularized* fashion, as classically done at the kinematic level, for instance in [15]. The resulting closed-form solution of the controller for a single elementary task is then: $\mathbf{u}_i=\mathbf{N}_i^{-1}\tilde{\mathbf{M}}_i^{\top}(\mathbf{I}\lambda_i^{-1}+\tilde{\mathbf{M}}_i\mathbf{N}_i^{-1}\tilde{\mathbf{M}}_i^{\top})^{-1}(\mathbf{b}_i+\tilde{\mathbf{M}}_i\mathbf{f})$, with $\tilde{\mathbf{M}}_i=\mathbf{A}_i\mathbf{M}^{-1}$. λ_i is a regularizing factor (we refer to [1] for a more accurate description of the regularization problem leading to this closed-form solution).

B. Controller for multiple elementary tasks with soft task priorities

Each elementary task is modulated by a task priority or task weight function $\alpha_i(t)$. To automatically find the optimal n_t task priorities $\{\alpha_i(t)\}_{i=1,\dots,n_t}$, we transform the functional optimization problem into a numerical optimization problem by representing the task priorities with parametrized functional approximators $\alpha_i(t)\rightarrow\hat{\alpha}_i(\hat{\boldsymbol{\pi}}_i,t)$, where $\hat{\boldsymbol{\pi}}_i$ is the set of parameters that shape the temporal profile of the i-th task

weight function. Following the scheme of Fig. 2, given n_t elementary tasks the final controller is given by:

$$\mathbf{u}(\mathbf{q},\dot{\mathbf{q}},t) = \sum_{i=1}^{n_t} \hat{\alpha}_i(\hat{\boldsymbol{\pi}}_i,t)\,\mathbf{u}_i(\mathbf{q},\dot{\mathbf{q}}) \ . \tag{1}$$

C. Learning the task priorities

We model the task priorities as a weighted sum of normalized Radial Basis Functions (RBFs):

$$\hat{\alpha}_i(\hat{\boldsymbol{\pi}}_i,t) = S\left(\frac{\sum_{k=1}^{n_r}\hat{\pi}_{ik}\psi_k(\mu_k,\sigma_k,t)}{\sum_{k=1}^{n_r}\psi_k(\mu_k,\sigma_k,t)}\right), \tag{2}$$

where $\psi_k(\mu_k,\sigma_k,t) = \exp\left(-1/2[(t-\mu_k)/\sigma_k]^2\right)$, with fixed mean μ_k and variance σ_k of the basis functions, n_r is the number of RBFs and $\hat{\boldsymbol{\pi}}_i = (\hat{\pi}_{i1},\ldots,\hat{\pi}_{in_r}) \subseteq \mathbb{R}^{n_P}$ is the set of parameters for each task priority. $S(\cdot)$ is a sigmoid function that squashes the output to the range $[0,1]$. The elementary task is fully activated when the task priority is equal to 1, otherwise the control action fades out until a full deactivation occurs when the priority goes to 0. The free parameters $\hat{\boldsymbol{\pi}}_i$ of each task weight function (Eq. 2) constitute the current parameters set to optimize: $\boldsymbol{\pi} = (\hat{\boldsymbol{\pi}}_1,\ldots,\hat{\boldsymbol{\pi}}_{n_t})$.

To optimize the free parameters $\boldsymbol{\pi}$, we introduce two elements, the fitness function ϕ and the set of inequality and equality constraints g,h:

- the fitness function $\phi = \phi(\mathbf{q}_{t=1,\ldots,T},\mathbf{u}_{t=1,\ldots,T},t)$ computes a performance measure of the global task executed by the robot over T time steps with the current parameters $\boldsymbol{\pi}$. The fitness function can contain different criteria ranging from energy consumption arguments to specific properties of the desired trajectories (*e.g.* speed and smoothness).
- the constraints g,h determine the admissible controls to be applied to the robot. They can be dependent on the robot structure (*e.g.* maximum joint torques and joint ranges), on the environment (*e.g.* obstacles and collisions), on the tasks (*e.g.* safety limits and couplings), *etc*.

The objective of the next section is to formalize the problem of optimizing the parameters $\boldsymbol{\pi}$ that maximize the fitness ϕ, ensuring that the constraints g,h are satisfied.

III. Constrained black-box optimization of task parameters

Learning the parameters $\boldsymbol{\pi} \in \boldsymbol{\Pi} \subseteq \mathbb{R}^{n_P}$ is a constrained optimization problem, as we need to find the optimal parameters $\boldsymbol{\pi}^\circ$ that maximize the objective function $J(\boldsymbol{\pi}) : \mathbb{R}^{n_P} \to \mathbb{R}$ (by default, equivalent to the fitness ϕ):

$$\boldsymbol{\pi}^\circ = \arg\max_{\boldsymbol{\pi}} J(\boldsymbol{\pi})$$

under the inequality and equality constraints g,h:

$$g_i(\boldsymbol{\pi}) \leq 0, i = 1,\ldots,n_{IC}; \quad h_j(\boldsymbol{\pi}) = 0, j = 1,\ldots,n_{EC} \ .$$

Following our approach in [1], we do not constrain the fitness structure nor its differentiability properties, hence we keep solving the problem with derivative-free methods. In [1] we used CMA-ES [10] for the known advantage of having to tune few parameters. To find feasible solutions that satisfy the constraints, we adopted a death penalty approach. This was clearly not efficient; the constant penalty applied to the fitness has a pathological effect on the exploration of the algorithm, possibly causing the search to get stuck in infeasible regions.

In this paper, we adopt a different strategy and look explicitly for variants of CMA-ES that take into account the constraints in the exploration procedure. Our goals are: 1) to improve the efficiency of the optimization procedure exploiting the constraint information, and 2) to guarantee that every solution found by the stochastic optimization process lies in a region of the parameter space that satisfies all the constraints. Interestingly, we are not interested in algorithms that permit constraints relaxation (hence violation) to find a solution: this is typically the case of real-time quadratic solvers (*e.g.* quadprog and qpOASES).

Among the multitude of constrained black-box optimization algorithms, we focused on three variants of CMA-ES: a *vanilla penalty* CMA-ES, the CMA-ES with *adaptive penalty* approach proposed in [12] and the (1+1)-CMA-ES with *covariance constrained adaptation* proposed in [2]. The first is a baseline CMA-ES that applies a penalty to the fitness that is proportional to the constraint violation. The second method is similar in principle, but the penalty weights are changed following a heuristic that depends on the constraint violation The third does not rely on penalties but updates the covariance whenever a constraint is violated, to drive the exploration away from infeasible regions.

In the rest of this section, we outline the three methods explaining their differences with respect to CMA-ES. In the presentation, we will use the following symbols:

- $J(\cdot)$: objective function
- n_{IC}: number of inequality constraints $g_i(\cdot)$
- n_{EC}: number of equality constraints $h_i(\cdot)$
- $n_C = n_{IC} + n_{EC}$: total number of constraints
- $\boldsymbol{\Pi} \subseteq \mathbb{R}^{n_P}$: parameter space
- $\boldsymbol{\pi}_k \in \boldsymbol{\Pi}$: k-th candidate at the current generation
- K: total number of candidates for each generation
- K_e: number of best candidates or *elites*
- $\boldsymbol{\pi}_{1:K_e}$: best candidates of the current generation
- $\mathcal{N}(\bar{\boldsymbol{\pi}},\boldsymbol{\Sigma})$: Gaussian distribution with mean $\bar{\boldsymbol{\pi}}$ and covariance $\boldsymbol{\Sigma}$
- σ^2: step size
- $l(\boldsymbol{\pi}_k)$: penalty factor
- $\hat{J}(\boldsymbol{\pi}_k) = J(\boldsymbol{\pi}_k) + l(\boldsymbol{\pi}_k)$: penalized objective function

A. Stochastic optimization with CMA-ES (no constraints)

A single iteration (called *generation*) of CMA-ES [10] consists of several steps. A set of K samples $\boldsymbol{\pi}_k$ is drawn from a multivariate Gaussian distribution $\mathcal{N}(\bar{\boldsymbol{\pi}},\sigma^2\boldsymbol{\Sigma})$ with a σ^2 step size; for each sample $\boldsymbol{\pi}_k$ we perform the evaluation of the objective function J, called *fitness*. The samples are sorted using a ranking procedure based on the fitness and an update of the Gaussian distribution is performed according to the best K_e candidates $\boldsymbol{\pi}_{1:K_e}$, called *elites*.

The update step affects the mean, covariance, and step size of the search distribution $\mathcal{N}(\bar{\boldsymbol{\pi}},\sigma^2\boldsymbol{\Sigma})$. The evolution of the mean is influenced by the probability weights P_k of each elite

CMA-ES without constraints

function CMA-ES
  for each $gen = 1,\ldots,n_{GENERATIONS}$ **do**
    for each $k = 1,\ldots,K$ **do**
      $\boldsymbol{\pi}_k \sim \mathcal{N}(\bar{\boldsymbol{\pi}}, \sigma^2\boldsymbol{\Sigma})$ ▷ **samples**
      $J_k = J(\boldsymbol{\pi}_k)$ ▷ **evaluation**
    end for
    $\boldsymbol{\pi}_{1:K_e} = \text{SORT}(\boldsymbol{\pi}_{k=1:K}, J_{k=1:K})$ ▷ **sorting**
    $\bar{\boldsymbol{\pi}}^{new} = \sum_{k=1}^{K_e} P_k \boldsymbol{\pi}_k$ with $\sum_{k=1}^{K_e} P_k = 1$
    $\boldsymbol{\Sigma}^{new} = \text{UPDCOV}(\bar{\boldsymbol{\pi}}^{new}, P_{k=1:K_e})$
    $\sigma^{new} = \text{UPDSIGMA}(\sigma)$
    $\bar{\boldsymbol{\pi}} = \bar{\boldsymbol{\pi}}^{new}$ $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{new}$ $\sigma = \sigma^{new}$
  end for
end function

Fig. 3. Pseudo-code for the basic CMA-ES without constraints.

candidate. A common choice is $P_k = \ln(0.5(K_e+1)) - \ln(k)$. In CMA-ES, premature convergence is avoided by tuning the step size σ^2. Both σ^2 and $\boldsymbol{\Sigma}$ are updated by combining the information from the last generation and all the previous ones. For the update of the stepsize σ^2 and more detail about the algorithm, we refer the reader to [10]. To initialize CMA-ES the user has to specify the exploration rate, a scalar value between [0,1] that controls the starting value of the covariance matrix. Fig. 3 shows the pseudo-code of the algorithm.

B. CMA-ES with Vanilla Constraints

The *vanilla penalty* functions method relies on adding a penalty term to the fitness of a candidate that depends on the constraints violation of the candidate. The method employs a penalized objective function $\hat{J}(\boldsymbol{\pi}_k) = J(\boldsymbol{\pi}_k) + l(\boldsymbol{\pi}_k)$ with the penalty factor $l(\boldsymbol{\pi}_k)$ defined as:

$$l(\boldsymbol{\pi}_k) = \sum_{i=1}^{n_{IC}} r_i \max(0, g_i(\boldsymbol{\pi}_k))^2 + \sum_{j=1}^{n_{EC}} c_j |h_j(\boldsymbol{\pi}_k)|$$

where r_i and c_i are positive constant values. In Fig. 4 we present a pseudo-code for this variant where we refer to the penalization routine with $\text{PENALTY}(\cdot)$.

C. CMA-ES with Adaptive Constraints

The previous method is by far the simplest and the most intuitive, as it applies a penalty that depends on the candidate $\boldsymbol{\pi}_k$. However, one may want to make the penalty term variable, for example depending on the exploration path.

Collange *et al.* [12] proposed a penalty function approach where a set of adaptive weights are tuned to prevent the search process from getting stuck in a local minima of the penalized fitness function $\hat{J}(\cdot)$. A penalized objective function $\hat{J}(\boldsymbol{\pi}_k)$ is therefore used. The key idea is that the penalty factor $l(\boldsymbol{\pi}_k)$ is built to consider the number of feasible solutions per each generation and the activation of each constraint, determined by a heuristic tuned by a user-defined ε_i. In particular, one assigns $l(\boldsymbol{\pi}_k) = \sum_{i=1}^{n_C} w_i [\gamma_i^+(\boldsymbol{\pi}_k)]^2$, where w_i, $i = 1,\ldots,n_C$ is the set of adaptive weights, and $\gamma_i^+(\cdot)$ is the positive part of the so-called *ε-normalized constraint values* γ_i, which are used to identify the active constraints. The ε-normalized constraint values γ_i are defined as:

$$\gamma_i = \begin{cases} [g_i(\boldsymbol{\pi}_k) + \varepsilon_i]/\varepsilon_i & \text{for inequality constraints} \\ |h_i(\boldsymbol{\pi}_k)|/\varepsilon_i & \text{for equality constraints} \end{cases} . \quad (3)$$

The user can tune the definition of the constraint violations and the relaxation of the constraints through the parameters $\varepsilon_i > 0, i = 1,\ldots,n_C$. For equality constraints $h_i(\cdot)$, a candidate solution $\boldsymbol{\pi}_k$ is labelled "feasible" if $0 \le \gamma_i \le 1$ and "infeasible" otherwise. For inequality constraints $g_i(\cdot)$, a candidate solution $\boldsymbol{\pi}_k$ is labelled "feasible" if $\gamma_i \le 1$ and "infeasible" otherwise (Fig. 5).

The weights update is driven by the ratio of feasible solutions for all the constraints:

$$r_i^{feas} = \frac{\#\text{ feasible solutions for the } i\text{-th constraint in the curr. generation}}{K}$$
$$\bar{r}_i^{feas} = \text{average of } r_i^{feas} \text{ over the last } n_P + 2 \text{ generation}$$

If all the samples $\boldsymbol{\pi}_k$ with $k = 1,\ldots,K$ satisfy the i-th constraint, then $r_i^{feas} = 1$; otherwise $r_i^{feas} < 1$. So $\bar{r}_i^{feas} = 1$ only when all the samples satisfy the i-th constraint for $n_P + 2$ generations. Once this condition is met, the weight w_i related to the i-th constraint is decreased almost surely (in a statistical sense). The adaptation rule for the weight w_i after each generation is defined as: $w_i = w_i \exp(p_{target} - r_i^{feas})$, where p_{target} is a value that changes at each iteration according to $p_{target} = (1/Kn_P)^{1/\mathscr{D}}$, where $\mathscr{D}$ represents the cardinality of the elements' set that satisfies $i = 1,\ldots,n_C : \bar{r}_i^{feas} < 1$ and K is the number of samples. As a rule of thumb, when $r_i^{feas} > p_{target}$ the weight w_i decreases, otherwise it increases. In Fig. 4 we provide a pseudo-code for this variant, where we refer to the weight computation routine as $\text{UPDATEWEIGHT}(\cdot)$. For more detail on the method we refer to [12].

This method is more interesting since the penalty factor applied to the objective function changes during the optimization process depending on the number of feasible solutions that do not violate the constraints, considering the relaxation acting on the equality constraints. With respect to the vanilla method, the penalty factor here is not constant over the parameter space and depends on the exploration path in the parameter space. This decreases the possibility of getting stuck in local minima or flat areas of the fitness.

D. (1+1)-CMA-ES with Covariance Constrained Adaptation

The third method, proposed by Arnold *et al.* [2], is an extension of (1+1)-CMA-ES with active covariance adaptation [16]. As opposed to the other two methods, here we do not have a penalty factor, *i.e.*, the objective function is unchanged, but there is a different exploration strategy that exploits the constraints information to change the covariance and keep the optimization in a feasible region.

A notable difference with the classical CMA-ES is the fact that there is only one sample per generation ($\boldsymbol{\pi}_1$, therefore $K = 1$), that is generated according to the following rule:

$$\boldsymbol{\pi}_1 = \boldsymbol{\pi} + \sigma \mathbf{D}\mathbf{z} \quad (4)$$

where $\mathbf{D}$ is the Cholesky factor of the covariance matrix $\boldsymbol{\Sigma} = \mathbf{D}^T\mathbf{D}$ and $\mathbf{z}$ is a standard normal distributed vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The algorithm stores the information about the successful steps in a *search path* $\mathbf{s} \in \mathbb{R}^{n_P}$. Each time a

CMA-ES with vanilla constraints

function CMA-ES
  for each $gen = 1,\dots,n_{GENERATIONS}$ **do**
    for $k = 1,\dots,K$ **do**
      $\boldsymbol{\pi}_k \sim \mathcal{N}(\boldsymbol{\pi}, \sigma^2\boldsymbol{\Sigma})$
    end for
    for $k = 1,\dots,K$ **do**
      $J_k = J(\boldsymbol{\pi}_k)$
      if ConstrViolation$(\boldsymbol{\pi}_k)$ **then**
        $\hat{J}_k = \text{Penalty}(\boldsymbol{\pi}_k, J_k)$
      end if
    end for
    $\boldsymbol{\pi}_{1:K_e} = \text{Sort}(\boldsymbol{\pi}_{k=1:K}, \hat{J}_{k=1:K})$
    $\boldsymbol{\pi}^{new} = \sum_{k=1}^{K_e} P_k \boldsymbol{\pi}_k$ with $\sum_{k=1}^{K_e} P_k = 1$
    $\boldsymbol{\Sigma}^{new} = \text{UpdCov}(\boldsymbol{\pi}^{new}, P_{k=1:K_e})$
    $\sigma^{new} = \text{UpdSigma}(\sigma)$
    $\bar{\boldsymbol{\pi}} = \bar{\boldsymbol{\pi}}^{new}$ $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{new}$ $\sigma = \sigma^{new}$
  end for
end function

CMA-ES with adaptive constraints

function CMA-ES
  for each $gen = 1,\dots,n_{GENERATIONS}$ **do**
    for $k = 1,\dots,K$ **do**
      $\boldsymbol{\pi}_k \sim \mathcal{N}(\boldsymbol{\pi}, \sigma^2\boldsymbol{\Sigma})$
    end for
    for $k = 1,\dots,K$ **do**
      $J_k = J(\boldsymbol{\pi}_k)$
    end for
    $[l, r, \bar{r}] = \text{CollectViolation}(\boldsymbol{\pi}_k, \boldsymbol{\varepsilon})$
    $w^{new} = \text{UpdateWeight}(w, r, \bar{r})$
    $\hat{J}_k = \text{WeightPenalty}(w^{new}, l)$
    $\boldsymbol{\pi}_{1:K_e} = \text{Sort}(\boldsymbol{\pi}_{k=1:K}, \hat{J}_{k=1:K})$
    $\boldsymbol{\pi}^{new} = \sum_{k=1}^{K_e} P_k \boldsymbol{\pi}_k$ with $\sum_{k=1}^{K_e} P_k = 1$
    $\boldsymbol{\Sigma}^{new} = \text{UpdCov}(\boldsymbol{\pi}^{new}, P_{k=1:K_e})$
    $\sigma^{new} = \text{UpdSigma}(\sigma)$
    $\bar{\boldsymbol{\pi}} = \bar{\boldsymbol{\pi}}^{new}$ $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{new}$ $\sigma = \sigma^{new}$
  end for
end function

(1+1)-CMA-ES with Cov. Const. Adapt.

function (1+1)-CMA-ES
  $\boldsymbol{\pi} = \text{FindFeasibleStartingPoint}()$
  for each $gen = 1,\dots,n_{GENERATIONS}$ **do**
    $\boldsymbol{\pi}_1 = \boldsymbol{\pi} + \sigma\mathbf{Dz}$ (Eq.4)
    if ConstrViolation$(\boldsymbol{\pi}_1)$ **then**
      $\mathbf{D}^{new} = \text{UpCovConstr}()$; $\mathbf{D} = \mathbf{D}^{new}$
    else
      $J^{new} = J(\boldsymbol{\pi}_1)$
      if $J^{new} > J$ **then**
        $\mathbf{D}^{new} = \text{UpCovSucc}()$
        $\sigma^{new} = \text{UpdSigma}(\sigma)$
        $\boldsymbol{\pi} = \boldsymbol{\pi}_1$; $\mathbf{D} = \mathbf{D}^{new}$; $\sigma = \sigma^{new}$
      else if $J^{new} > J^{old}$ **then**
        $\mathbf{D}^{new} = \text{UpCovActive}()$; $\mathbf{D} = \mathbf{D}^{new}$
      end if
    end if
  end for
end function

Fig. 4. Pseudocode for the three variants of constrained CMA-ES: the first is with vanilla penalty (Section III-B), the second is the adaptive penalties method of [12] (Section III-C) and the third is a (1+1)-CMA-ES with covariance constrained adaptation as in [2] (Section III-D).

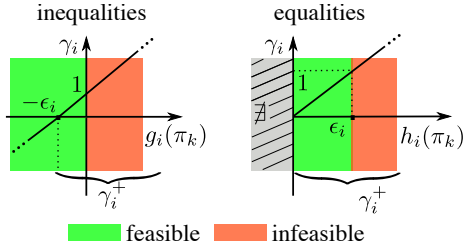

Fig. 5. This illustration shows the relation between ε_i and γ_i for inequality and equality constraints as in Eq. 3. As described in Section III-C, the green and red regions identify the constraint values that are respectively labeled as "feasible" and "infeasible". One may notice that ε_i induces a relaxation for the equality constraint: therefore it could be possible to label as feasible a solution that violates the constraint (how much depends on ε). On the contrary, it is noticeable that γ_i^+ in the inequality constraint also includes a boundary region determined by ε_i where the constraint is satisfied.

candidate outperforms the current best, $\mathbf{s}$ and $\mathbf{D}$ are updated (UpCovSucc in Fig. 4):

$$\mathbf{s}^{new} = (1-c)\mathbf{s}\sqrt{c(2-c)}\mathbf{Dz}$$

$$\mathbf{D}^{new} = \sqrt{1-c_{cov}^+}\mathbf{D} + \frac{\sqrt{1-c_{cov}^+}}{||\mathbf{w}||^2}\left(\sqrt{1+\frac{c_{cov}^+||\mathbf{w}||^2}{1-c_{cov}^+}}-1\right)\mathbf{sw}^T$$

where c_{cov}^+ and c are both factors that control the update rate of $\mathbf{s}$ and $\mathbf{D}$ respectively, while $\mathbf{w} = \mathbf{D}^{-1}\mathbf{s}$. Instead, if the current candidate is feasible but its performance is lower than the predecessors, the Cholesky factor $\mathbf{D}$ is actively updated (UpCovActive in Fig. 4):

$$\mathbf{D}^{new} = \sqrt{1+c_{cov}^-}\mathbf{D} + \frac{\sqrt{1+c_{cov}^-}}{||\mathbf{z}||^2}\left(\sqrt{1-\frac{c_{cov}^-||\mathbf{z}||^2}{1+c_{cov}^-}}-1\right)\mathbf{Dzz}^T$$

where c_{cov}^- is again a constant that determines the update rate. In this case $\mathbf{s}$ is not updated because the current candidate is not better in terms of fitness.

To handle constraints, the key idea is to update the covariance matrix, by reducing the components of $\mathbf{Dz}$ in the direction that is orthogonal to the constraint whenever a constraint is violated, as illustrated in Fig. 6. Each time the j-th constraint is violated, we update the corresponding constraint vector $\mathbf{v_j} \in \mathbb{R}^{n_P}$ and the matrix $\mathbf{D}$ (UpCovConstr in Fig. 4):

$$\mathbf{v}_j^{new} = (1-c_c)\mathbf{v}_j + c_c\mathbf{Dz}$$

$$\mathbf{D}^{new} = \mathbf{D} - \frac{\beta}{\sum_{j=1}^{n_C}\mathbf{1}_{g_j(\boldsymbol{\pi}_1>0)}}\sum_{j=1}^{n_C}\mathbf{1}_{g_j(\boldsymbol{\pi}_1>0)}\frac{\mathbf{v}_j\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}}$$

where c_c and β are constants that tune the update step respectively for $\mathbf{v}_j$ and $\mathbf{D}$, $\mathbf{w}_j = \mathbf{D}^{-1}\mathbf{v}_j$ and $\mathbf{1}_{g_j(\boldsymbol{\pi}_1>0)}$ is equal to one when $g_j(\boldsymbol{\pi}_1) > 0$ and zero otherwise.

In summary, the method searches for the optimal solution by testing one sample at the time and accounting for the constraints in the covariance adaptation to stay away from infeasible regions. The algorithm is designed in such a way that the mean of the search distribution is updated only if the fitness improves and the candidate is a feasible solution; these two elements ensure that the solution of the optimization problem always satisfies the constraints. However, unlike the other methods, this requires a feasible[1] starting candidate to work, otherwise the exploration process quickly gets stuck. Hence, this method cannot be started from scratch or random values, but needs the pre-computation of a feasible starting point. This is not an issue, since we can always find a feasible starting point that satisfies all the constraints, even if it does not enable the robot to achieve the global task goal (a quick solution is to set the robot in a feasible posture and keep this position by setting the posture task priority to 1 and the others to 0).

IV. Benchmarking the Algorithms

In this section we test the algorithms described in Section III to decide which one better suits our problem. We compare their performances on five different benchmarks:

[1]A candidate solution is feasible if it satisfies all the constraints.

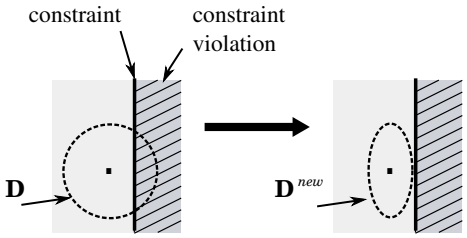

Fig. 6. This illustration shows the effect of the covariance adaptation with constraints, as described in Section III-D. A linear inequality constraint, represented by the vertical thick line, divides the parameter space into a region where the constraint is not violated (light grey) and a region where the constraint is violated (dark grey). The covariance $\mathbf{D}$ of the search distribution is updated in such a way that the successor samples will not fall into the region where the constraint is active: the updated covariance $\mathbf{D}^{new}$ is directed orthogonally with respect to the constraint.

- g07: $n_P = 10, n_{IC} = 8, n_{EC} = 0$
- g09: $n_P = 7, n_{IC} = 4, n_{EC} = 0$
- HB: $n_P = 5, n_{IC} = 6, n_{EC} = 3$
- RB1: $n_P = 15, n_{IC} = 32, n_{EC} = 0$
- RB2: $n_P = 15, n_{IC} = 50, n_{EC} = 0$

The first three are classical benchmarks for constrained optimization [2], that is analytic problems with known optimal solutions; the last two are new benchmarks that we designed *ad hoc* to evaluate the performance of the algorithms on robotic problems with growing complexity. RB1 is a problem inspired by our previous work [1] where a KUKA LWR (7DOF) has to reach a goal position with its end-effector behind an obstacle, while satisfying constraints of joint position limits, joint torque limits and obstacle avoidance. RB2 has a similar setting with the addition of a second obstacle to avoid and another set of constraints coming from joint velocity limits. To compare the performance of the algorithms on these benchmarks, we define the following metrics:

- m_1: *distance from the optimal solution*, defined as $m_1 = \|\boldsymbol{\pi}^\circ - \boldsymbol{\pi}^*\|$, where $\boldsymbol{\pi}^\circ$ is the optimal solution (known) and $\boldsymbol{\pi}^*$ the best solution found by the constrained optimization algorithm;
- m_2: *constraint violations*, defined as $m_2 = \sum_{i=1}^{n_C} |\hat{e}(i, \boldsymbol{\pi}^*)|$, where $\hat{e}(i, \boldsymbol{\pi}) = \mathbf{1}_{g_i(\boldsymbol{\pi})>0}\, g_i(\boldsymbol{\pi})$ for the inequality constraints and $\hat{e}(i, \boldsymbol{\pi}) = \mathbf{1}_{h_i(\boldsymbol{\pi})\neq 0}\, h_i(\boldsymbol{\pi})$ for the equality constraints — basically it sums all the constraints that are violated;
- m_3: *number of steps to converge*, or *settling time*, defined as $m_3 = n_{sc}(\delta)$, the number of steps after which the fitness function reaches a steady state condition, *i.e.*, its value is bounded between $\pm\delta\%$ of the steady state value — here, we set $\delta = 2.5$;
- m_4: *best fitness*, defined as $m_4 = J(\pi^*)$, *i.e.*, the fitness of the best solution found by the constrained optimization algorithm.

To provide a baseline, we use the (deterministic) constrained optimization function *fmincon* in Matlab, using the SQP method. This is a suitable choice because it does not require the gradient of the objective function for non-linear constrained optimization problem with nonlinear constraints.

Since (1+1)-CMA-ES with covariance constrained adaptation (Section III-D) needs a feasible candidate solution as a starting point, in order to make a fair comparison all the algorithms start from the same initial position. We perform 40 repetitions of the optimization process per each test problem for each algorithm with an exploration rate of 0.1 and a 5000 samples to assure the convergence of the methods.

Fig. 7 shows the results of the numerical experiments with the five benchmarks. The top row reports on the results for g07, g09 and HB with metrics m_1, m_2, m_3, while the bottom row reports on the results for the robotics benchmarks RB1 and RB2, with metrics m_2, m_3, m_4 (m_1 cannot be used in this case because the optimal solution $\boldsymbol{\pi}^\circ$ is not known). We also compared the four algorithms in terms of computational time, and did not find significant differences (for example, the optimal solution for RB2 is found on average in $\approx$1.7e+04 *s* for the CMA-ES variants and 1.9e+04 *s* for *fmincon* on a i5 laptop with Matlab).

(1+1)-CMA-ES with covariance constrained adaptation offers the best trade-off between performance and constraints' satisfaction both on the analytic and the robotic benchmarks. It always ensures full satisfaction of the constraints while the other methods sometimes fail. Its settling time is comparable to the other stochastic algorithms, while *fmincon* converges faster. *fmincon* could seem more appealing, but on the robotic benchmarks its best fitness is lower and actually quite close to the fitness of the starting point (meaning that the algorithm does not really "explore"). Therefore *fmincon* does not seem a suitable candidate for solving robotic problems with a lot of constraints.

The different performances of the algorithms in the analytic and robotic benchmarks confirm the benefit gained by designing two new robotics benchmarks RB1,RB2. Overall, considering the zero constraint violations and the capability of finding a good solution, we choose to use (1+1)-CMA-ES with covariance constrained adaptation for our experiments with the iCub robot.

V. Robotic experiments

In this section, we apply (1+1)-CMA-ES with covariance constrained adaptation to our multi-task control framework (Section II). We use it to optimize the task priorities and to obtain a solution that never violates the constraints. In the following, we report on the experiments performed to optimize the whole-body movements of the iCub humanoid robot.

We designed two experiments using the 17 DOF of the upper-body of the robot (arms and torso). In the experimental scenario, a rectangular obstacle similar to a wall, that is as large as the robot's chest and 2 *cm* thick, is placed about 20 *cm* in front of the robot.

The first experiment is aimed at reaching a goal Cartesian position behind the wall with one hand. There are three elementary tasks. The first is about reaching the desired Cartesian position $\mathbf{p}_r^* = [0.35, -0.15, 0.7]$ (m) with the right hand frame of the robot. The second task is reaching a desired

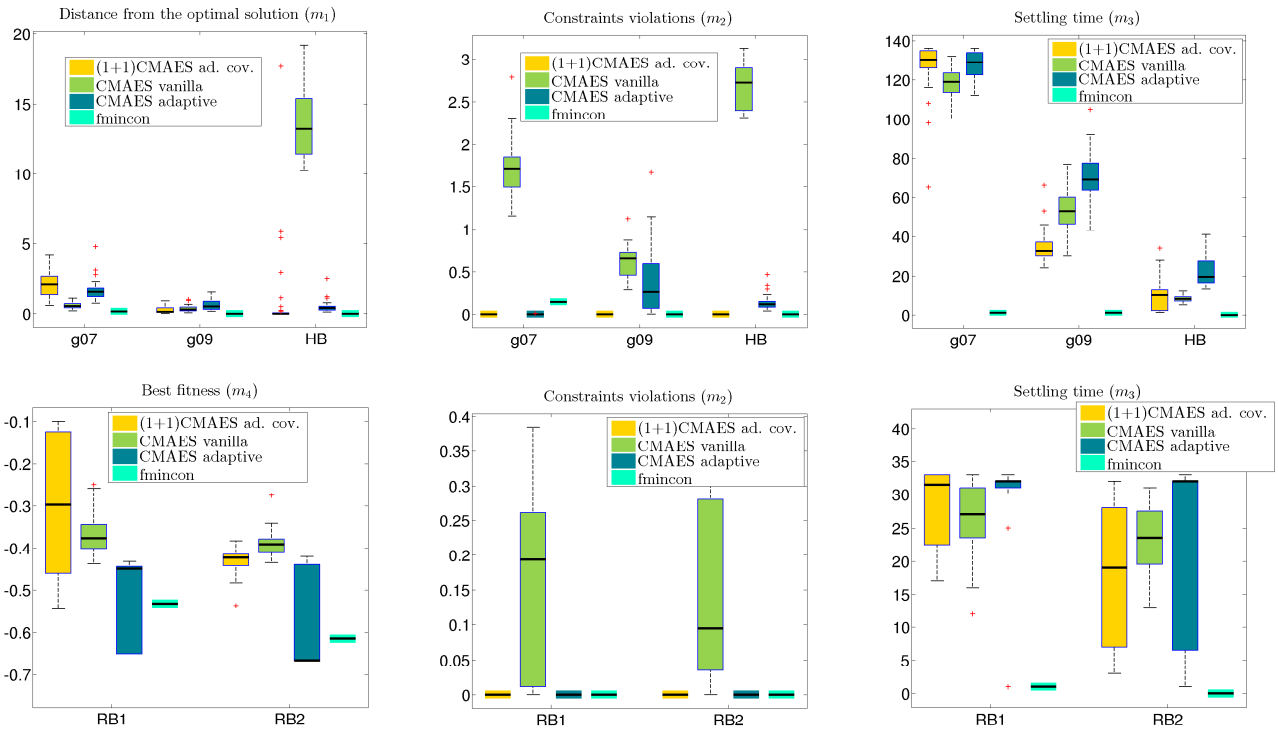

Fig. 7. Performance comparison of the three constrained CMA-ES algorithms and the baseline *fmincon* algorithm from Matlab using the SQD method. The top row reports on the results on three standard analytical constrained optimization benchmarks (g07, g09, HB - see [2]). The bottom row reports on the results on two robotics benchmarks (RB1, RB2) that we designed *ad hoc* to evaluate the performance of the algorithms on robotics problems.

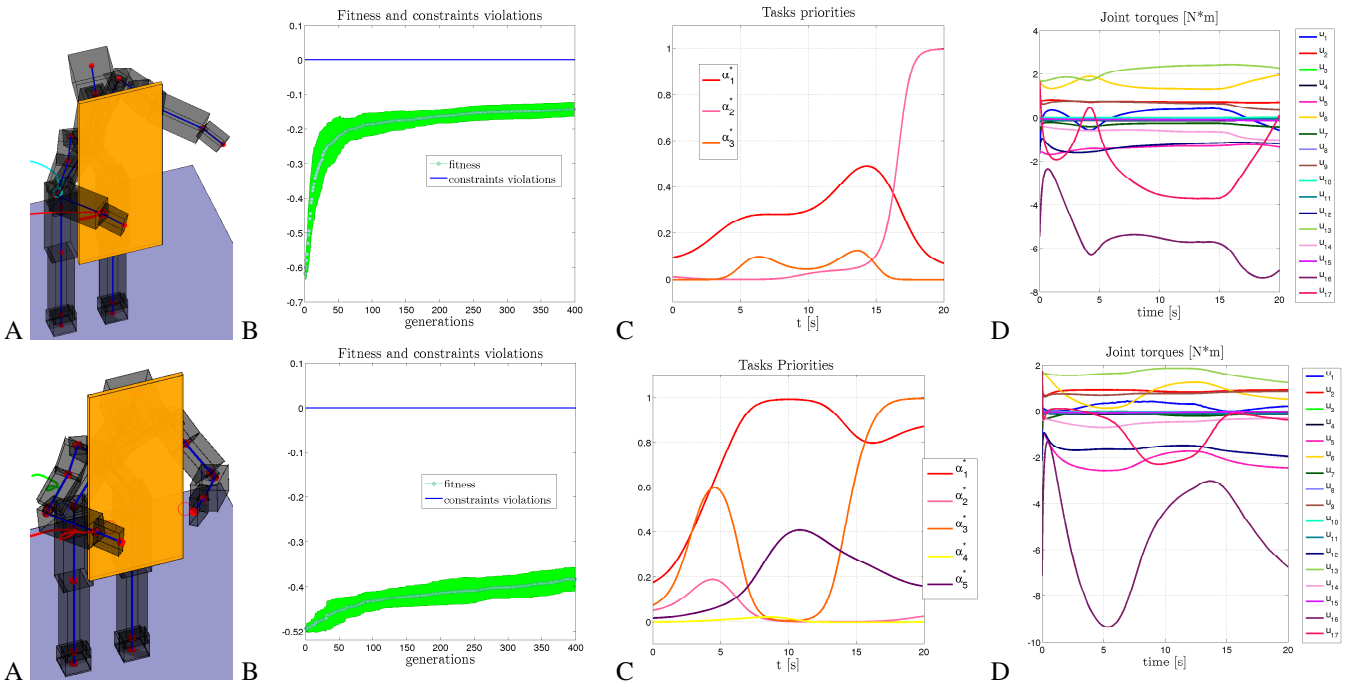

Fig. 8. Two experiments with the iCub, about reaching a goal behind the wall with one or two hands. A) The robot's movement visualized by the mex model. B) The median constraint violation and fitness optimized by (1+1)-CMA-ES with covariance constrained adaptation (over 25 experiments) the constraints are never violated C-D. The task priorities and joint torques of the best solution. The experiments are also shown in the attached video.

Cartesian position $\mathbf{p}^*_{elbr} = [0.24, -0.23, 0.7]$ (m) with the elbow frame. The third task is keeping the initial joint configuration $\mathbf{q}^* = [0,45,0,0,-20,30,0,0,45,0,0,0,30,0,0,0,0]$ (deg). In sum, the goal is hidden behind the wall, and to

reach it with the hand the robot must bend its elbow around the wall corner; the third task should prevent the robot from moving the right arm and the torso. The task priorities are approximated by RBFs with $n_r = 5$, therefore $n_P = 5 \times 3 = 15$. There are $n_C = n_{IC} = 73$ inequality constraints: joint position limits, joint torque limits and distance constraints to avoid collisions between the robot and the obstacle. Precisely, a minimal distance of 3 *cm* is required between the obstacle and a set of pre-defined collision check points (located at the origin of the frames of right shoulder, elbow, wrist, hand and head). For this experiment we use the following fitness function:

$$\phi = -\frac{1}{2}\left[\frac{\sum_i^T \|\mathbf{p}_{r,i} - \mathbf{p}_r^*\|}{\varepsilon_{max}} + \frac{\sum_i^T \mathbf{u}_i^2}{u_{max}}\right] \quad (5)$$

where $\phi \in [-1, 0]$, T is the number of control steps (the task duration is 20 *s*, and we control at 1 *ms*), $\mathbf{p}_{r,i}$ is the right hand frame position at time i, $\mathbf{p}_r^*$ the goal position for the hand frame and $\varepsilon_{max} = 120$ and $u_{max} = 3.5 * 10^5$ are two scaling factors. The first term of ϕ penalizes the cumulative distance from the goal, while the second term penalizes the global control effort.

The second experiment complicates the first by adding 2 more tasks. The aim is to reach a Cartesian goal position with both robot hands. Two Cartesian goal tasks for each hand and elbow are set symmetrically with respect to iCub's sagittal plane. A fifth posture task is set as to keep the torso as straight as possible during the movement.

- Task 1 : $\mathbf{p}_r^* = [0.35, -0.15, 0.68]$ (m)
- Task 2 : $\mathbf{p}_{elbr}^* = [0.21, -0.25, 0.68]$ (m)
- Task 3 : $\mathbf{p}_l^* = [0.3, 0.0248, 0.68]$ (m)
- Task 4 : $\mathbf{p}_{elbl}^* = [0.21, 0.1138, 0.68]$ (m)
- Task 5 : $\mathbf{q}^* = [0, 45, 0, 0, -20, 30, 0, 0, 45, 0, 0, 0, 30, 0, 0, 0, 0]$ (deg)

The task priorities are approximated by RBFs with $n_r = 5$, therefore $n_P = 5 \times 5 = 25$. The optimization is carried out under the same constraints as in the first experiment with the addition of the left arm collision checks. This means we have $n_C = n_{IC} = 77$ inequality constraints. The fitness is:

$$\phi = -\frac{1}{2}\left(\frac{\sum_i^T \|\mathbf{p}_{r,i} - \mathbf{p}_r^*\|}{\varepsilon_{max}} + \frac{\sum_i^T \|\mathbf{p}_{l,i} - \mathbf{p}_l^*\|}{\varepsilon_{max}} + \frac{\sum_i^T \mathbf{u}_i^2}{u_{max}}\right) \quad (6)$$

where $\mathbf{p}_{l,i}$ is the left hand frame position at time i, $\mathbf{p}_l^*$ the goal position for the left hand frame.

In all the experiments, we seek the best solutions that do not violate any of the constraints. We employ (1+1)-CMA-ES as described in Section III-D with the exploration rate set to 0.1 (this is the only parameter to tune and this is the default value!).

Fig. 8 shows the median fitness and constraint violation obtained by 25 experiments. The fitness grows nicely ($\phi = 0$ would be the optimum). Most importantly, the **constraints are never violated**, which is exactly what we wanted to obtain. We also show the task priorities and the joint torques from one of the best solutions; they are both smooth, and it is clear that optimizing the task priorities manually would be very difficult if these solutions were to be achieved. The video attachment shows the robot movements in the two experiments and the activation of the tasks priorities evolving in time.

VI. Conclusion and Future Work

In this paper we proposed to optimize the task priorities of multi-task controllers by a stochastic constrained optimization algorithm that ensures that the constraints are never violated. We benchmarked four constrained optimization algorithms in robotics applications and found that (1+1)-CMA-ES with covariance constrained adaptation meets our requirements in terms of fitness of the solution and constraint satisfaction. Our framework can be used to generate optimized whole-body movements that always comply with safety requirements, as shown in two bimanual experiments with the iCub. Our current limit is the computation time, therefore the method is suited at this time only for offline synthesis of whole-body behaviors of humanoid robots. Ongoing work is aimed at applying the method for safe trajectory optimization (complementary to task priority optimization) and speeding up the computation.

References

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

[2] D. V. Arnold and N. Hansen, "A (1+1)-CMA-ES for constrained optimisation," in *GECCO*, 2012, pp. 297–304.

[3] L. Saab, O. Ramos, F. Keith, and N. Mansard et al, "Dynamic whole-body motion generation under rigid contacts and other unilateral constraints," *IEEE Trans. on Robotics*, vol. 29, pp. 346–362, 2013.

[4] 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 Auton. Systems*, vol. 63, pp. 150–157, 2015.

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

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

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

[8] S. Ha and C. Liu, "Evolutionary optimization for parameterized whole-body dynamic motor skills," in *ICRA*, 2016.

[9] R. Lober, V. Padois, and O. Sigaud, "Variance modulated task prioritization in whole-body control," in *IROS*, 2015.

[10] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies." *Evolutionary Computation*, vol. 9, pp. 159–195, Jan 2001.

[11] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with Gaussian processes," in *ICRA*, 2016.

[12] G. Collange, N. Delattre, N. Hansen, I. Quinquis, and M. Schoenauer, "Multidisciplinary optimization in the design of future space launchers," in *Multidisciplinary Design Optimization in Computational Mechanics*. Wiley-Blackwell, 2013, pp. 459–468.

[13] J. Peters, M. Mistry, F. Udwadia, J. Nakanishi, and S. Schaal, "A unifying framework for robot control with redundant DOFs," *Autonomous Robots*, vol. 24, pp. 1–12, Jan 2008.

[14] S. Chiaverini, B. Siciliano, and O. Egeland, "Redundancy resolution for the human-arm-like manipulator," *Robotics and Autonomous Systems*, vol. 8(3), pp. 239–250, Jan 1991.

[15] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *J. Dyn. Sys., Meas., Control*, vol. 108 (3), pp. 163–171, 1986.

[16] C. Igel, T. Suttorp, and N. Hansen, "A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies," in *GECCO*, 2006.