# Optimizing Task Feasibility using Model-Free Policy Search and Model-Based Whole-Body Control

Ryan Lober, Jorhabib Eljaik, Gabriele Nava, Stefano Dafarra, Francesco Romano, Daniele Pucci, Silvio Traversaro, Francesco Nori, Olivier Sigaud, Vincent Padois

# Optimizing Task Feasibility using Model-Free Policy Search and Model-Based Whole-Body Control

Ryan Lober[1], Jorhabib Eljaik[1], Gabriele Nava[2], Stefano Dafarra[2], Francesco Romano[2], Daniele Pucci[2], Silvio Traversaro[2], Francesco Nori[2], Olivier Sigaud[1], and Vincent Padois[1]

***Abstract*—Producing feasible motions for highly redundant robots, such as humanoids, is a complicated and high-dimensional problem. Model-based whole-body control of such robots, can generate complex dynamic behaviors through the simultaneous execution of multiple tasks. Unfortunately, tasks are generally planned without close consideration for the underlying controller being used, or the other tasks being executed, and are often infeasible when executed on the robot. Consequently, there is no guarantee that the motion will be accomplished. In this work, we develop an optimization loop which automatically improves task feasibility using model-free policy search in conjunction with model-based whole-body control. This combination allows problems to be solved, which would be otherwise intractable using simply one or the other. Through experiments on both the simulated and real iCub humanoid robot, we show that by optimizing task feasibility, initially infeasible complex dynamic motions can be realized — specifically, a sit-to-stand transition. These experiments can be viewed in the accompanying video.**

***Index Terms*—policy search, whole-body control, humanoids**

## I. Introduction

Highly redundant robots, such as humanoids, have enormous potential industrial and commercial utility. Unfortunately producing feasible and useful behaviors on real robots is a challenging undertaking, particularly when the robot must interact with the environment. This is caused, in large part, by the fact that there are always errors between what is planned, or simulated, and what is executed on a real robot due to modeling errors and perturbations. Consequently, an automatic method of resolving these errors on real platforms is absolutely necessary for robots to attain true autonomy. Model-based control alone cannot resolve these issues because the many possible causes could not be practically modeled for a general case. Similarly, even the most sample efficient end-to-end learning methods (e.g. [1]) would also fail because training a model on a real robot would require an inordinate number of evaluations, or rollouts. In this study, we show that by combining control and learning techniques, we can create low-dimensional high-level abstractions of whole-body behaviors and efficiently correct initially infeasible motions on real robots.

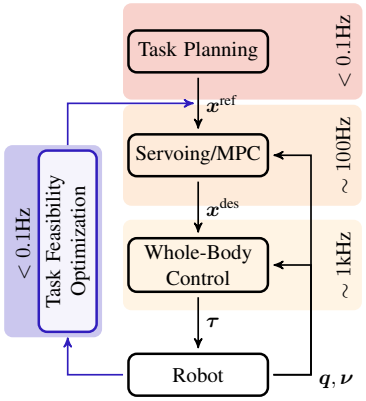


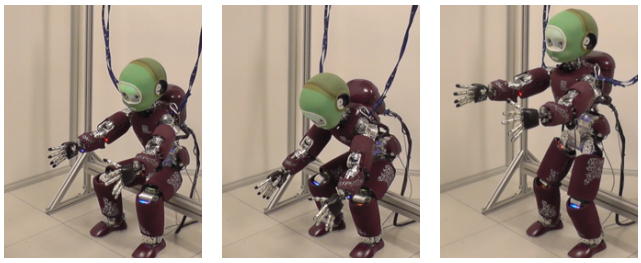

Fig. 1: A modern control hierarchy for highly redundant robotic systems, e.g. humanoid robots. At the lowest level is whole-body control, which determines the torques needed to accomplish a set of tasks. At the intermediate level, these tasks are controlled by the servoing/MPC level where task trajectory errors are compensated using feedback. Finally the task trajectories are provided by high-level planning, which is usually a combination of operator expertise and automated planning. The task feasibility optimization loop proposed in this paper is designed to correct infeasible tasks produced by this architecture.

Modern control architectures employ multiple control levels in order to decouple complex behaviors into manageable control problems. At the lowest level is *reactive whole-body control*, where joints torques are calculated at high frequency ($\sim$ 1kHz) given one or more tasks [2]. The control problem can be written as a constrained convex optimization, where the objective function is a combination of task errors, and the constraints are the equations of motion, articulation and actuation limits, and contacts [3], [4], [5]. Task errors are dictated by desired task values which come from the next level of *task servoing*. At this level, closed loop controllers are used to servo task trajectories using state feedback (PID) and/or Model Predictive Control (MPC) schemes at frequencies between 100Hz and 10Hz [6], [7]. These task trajectories generate the reference values, which are used by task servoing, and come from the higher-level *open-loop planning* which takes seconds to minutes, and generally combines operator expertise and automated planning algorithms [8], [9]. This control hierarchy of planning, servoing, and whole-body control is presented in Fig. 1.

Because the control problem is abstracted in the task

[1] The authors are with - Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005, Paris, France - CNRS, UMR 7222 , Institut des Systèmes Intelligents et de Robotique, F-75005, Paris, France
e-mail: firstname.lastname@isir.upmc.fr

[2] The authors are with - iCubFacility, Istituto Italiano di Tecnologia, Genoa, Italy,
e-mail: firstname.lastname@iit.it



(a) 0.0s (b) 3.0s (c) 8.0s

Fig. 2: In (a), (b), and (c), we show a time-lapse of a feasibility-optimized standing motion executed on an iCub robot.

servoing and planning levels, there is no guarantee that the task trajectories will be executed properly by the lower control layers. Furthermore, tasks may conflict with one another and/or the system constraints [10], [11]. The end result is typically unstable or undesirable whole-body behaviors, and we qualify these tasks as *infeasible*. Prioritization techniques are used to manage perturbations engendered by infeasible tasks at the whole-body control level, but are difficult to tune and only circumvent the problem. Moreover, tasks infeasibilities change over the course of the movement so applying static priorities may be overly restrictive [12], [13]. Likewise, tuning/scheduling the task servoing gains not only modifies the task trajectories, but also changes the controller's impedance, which may be undesirable for the application. Hence, decoupling the impedance problem from the trajectory shaping problem is not only prudent, but simplifies each because well designed task trajectories should alleviate the need for priority and gain tuning.

Given that it is the task reference values which generate the infeasible control solutions, the task trajectories must be altered. To do so, the errors induced by infeasibilities can be measured and the task trajectories may be modified to reduce them. Additionally, the servoing and whole-body control levels with all of their parameters, as well as the robot's dynamics and environment, need to be taken into account. Given the complexity of these requirements, it is impractical to analytically model the relationship between task trajectories and feasibility. One solution is therefore to use model-free policy search (PS) techniques to modify the trajectories through trial and error by minimizing a cost function [14].

The objective of this study is to establish the task feasibility optimization loop, shown on the left in Fig. 1, by iteratively improving task trajectories using PS and exploiting the model-based control layers. Building on the work in [15], we first formalize the relationship between task trajectories and parameterized policies in the whole-body control architecture. We then develop a task feasibility cost, the penalty function, from simple principles which measure the infeasibility of a task. This feasibility cost is then minimized. In robotics, it is advantageous, from both a time and monetary standpoint, to perform PS with the fewest possible rollouts. To this end, we use Bayesian Optimization (BO) for its sample efficiency. BO solvers usually require fewer trials to obtain an optimal solution and have become a popular choice in robotics because of this efficacy [16], [17], [18], [19].

To study task feasibility optimization, we explore the dynamically challenging activity of moving from sitting to standing for the humanoid robot iCub, both in simulation and on the real robot. This motion requires contact switching and potentially unstable dynamic equilibrium to succeed. In addition to a postural impedance task, a Center of Mass (CoM) task is used to manage the sit-to-stand transition. The trajectory of the CoM task is optimized to minimize the task feasibility cost. Through these experiments, we demonstrate that by combining analytical model-based controllers with data-driven model-free PS techniques, we are able to solve problems which would be otherwise intractable using simply one or the other — e.g. producing feasible dynamically complex motions on real robots, like the example shown in Figs. 2a-2c.

## II. Methods

In this section, we describe the methods and tools used to develop task feasibility optimization. We begin with an overview of the underlying whole-body control architecture and conclude with a description of PS. Here the policy to be optimized is parameterized by the CoM task trajectory.

### A. Control Architecture

Model-based whole-body controllers determine at each control instant, $k$, the joint torques, $\boldsymbol{\tau}(k)$, necessary to accomplish some set of $n_T$ tasks, for all of the degrees of freedom of the given robot, while respecting physical constraints such as the equations of motion, articulation and actuation limits, and contacts. These controllers can be formulated using analytical null-space projection methods [20], or multicriterion convex optimization problems using weighted [3], [4] and/or hierarchical objective scalarization [21]. Regardless of the formalism, any of these controllers can be abstracted to the following generic function,

$$\boldsymbol{\tau}(k) = \text{controller}\left(\boldsymbol{s}(k), \mathcal{C}(k), T_i(k)\right) \quad \forall i \in \{1, 2, \ldots, n_T\}\,, \tag{1}$$

which takes the robot's state, $\boldsymbol{s}(k)$, its constraints, $\mathcal{C}(k)$, and some tasks $T_i(k)$, as inputs and outputs the joint torques. The robot state, contains $\boldsymbol{q}(k)$, the generalized coordinates, and $\boldsymbol{\nu}(k)$, its derivative. The variable $\mathcal{C}(k)$ contains any active constraints, e.g. joint and actuator limits, contacts, etc. Tasks may be described in any number of ways in either operational-space or joint-space, but all are governed by desired task values provided by task servoing.

In an earlier version of this method, presented in [15], the whole-body controller described in [3] is used. In this work, the whole-body control algorithm used is the momentum-based hierarchical controller developed in [22], [23], which has momentum tracking, $T_{\text{m}}$, and joint impedance tasks, $T_{\text{j}}$, — the most important of which is the former. Equation (1) can then be written,

$$\boldsymbol{\tau}(k) = \text{controller}\left(\boldsymbol{s}(k), \mathcal{C}(k), T_{\text{m}}, T_{\text{j}}\right)\,. \tag{2}$$

For the momentum task, the desired value is entirely determined by the desired CoM acceleration, $\ddot{\boldsymbol{x}}^{\text{des}}_{\text{CoM}}$, and is provided by a proportional-integral servoing controller,

$$\ddot{\boldsymbol{x}}^{\text{des}}_{\text{CoM}} = \ddot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}} - K_p(\dot{\boldsymbol{x}}_{\text{CoM}} - \dot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}) - K_i(\boldsymbol{x}_{\text{CoM}} - \boldsymbol{x}^{\text{ref}}_{\text{CoM}}) \ , \quad (3)$$

where $K_p$ and $K_i$ are the proportional and integral gain matrices respectively. The CoM reference values, $\boldsymbol{x}^{\text{ref}}_{\text{CoM}}$, $\dot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}$, and $\ddot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}$ are provided by a CoM trajectory. The choice of this reference is thus crucial for a successful whole-body motion, and without it the controller would serve little purpose.

In the context of the sit-to-stand example explored here, a finite-state-machine (FSM) composed of two states, coordinates the standing motion in the controller. In the "Sit" state, the robot is seated on the bench, and the two contacts at the left and right upper legs are controlled to keep the equilibrium. When a resultant ground reaction force greater than 150N is detected, the FSM switches to the "Stand" state, moving the bench contacts to the left and right heels in the whole-body controller.

### *B. States, Actions, and Policies*

Policy search methods are black-box optimization techniques for iteratively learning control policies rather than programming them by hand [24]. Model-free parameterized PS lends itself to robotics as it precludes the need for an analytical transition dynamics model and allows high-dimensional problems to be handled with few parameters. In keeping with reinforcement learning nomenclature, we define the agent of this system, the humanoid robot (iCub), and its discrete-time states are $\boldsymbol{s}(k)$. The actions of the agent, $\boldsymbol{a}(k)$, are then the actuator torques, developed at each control instant, $\boldsymbol{a}(k) = \boldsymbol{\tau}(k)$. The control policies, $\boldsymbol{\pi}\left(\boldsymbol{a}(k)|\boldsymbol{s}(k)\right)$, determine the action at time $k$ given the current state. The policies are mappings from task reference inputs, $\boldsymbol{x}^{\text{ref}}_i$, $\dot{\boldsymbol{x}}^{\text{ref}}_i$, and $\ddot{\boldsymbol{x}}^{\text{ref}}_i$ $\forall i \in \{1, 2, \ldots, n_T\}$, to $\boldsymbol{\tau}$, using the whole-body reactive controller described in Sec. II-A. It should be noted that this mapping is not bijective and cannot be described by a differentiable function. Assuming fixed whole-body controller parameters, we can consider that the mapping depends only on $\boldsymbol{s}(k)$ and the task control objectives at each time step. Therefore, in order to modify $\boldsymbol{\pi}(k)$ we must modify the task reference values, i.e. the task trajectories.

### *C. Policy Parameterization: Task Trajectories*

Given the high dimensionality of the system's states and actions, we opt for a parameterized policy representation. As presented in Sec. II-B, task trajectories uniquely determine the evolution of the system, and therefore provide a condensed representation of $\boldsymbol{\pi}$ for a given motion. The task trajectories, and hence $\boldsymbol{\pi}$, are parameterized by a series of keyframes/waypoints, which represent task coordinates of particular importance. A single position waypoint is given by $\boldsymbol{\theta}_i$, while a set of $n_\theta$ waypoints is denoted $\Theta = \begin{bmatrix}\boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \ldots & \boldsymbol{\theta}_{n_\theta}\end{bmatrix}$. From $\Theta$, a policy must be formed using a parameterized function, $\boldsymbol{\pi}_\theta = \boldsymbol{\rho}(\Theta)$, where the $\boldsymbol{\rho}(\Theta)$ function can be chosen from a variety of parameterized trajectory generators: e.g. splines, polynomials, optimal control methods, etc. Here, we use the formulation proposed by [25], which produces a time-optimal trajectory through $\Theta$, with a duration, $t_{\boldsymbol{\pi}}$, dependent on the velocity and acceleration limits imposed on the movement. For this study, we focus on the CoM task trajectory, which will guide the robot from a seated state to a standing state and therefore write the policy as, $\boldsymbol{\pi} = \boldsymbol{\rho}(\Theta^{\text{CoM}})$, where $\Theta^{\text{CoM}}$ are the CoM waypoints. Note that any task trajectories can be used in the parameterization of $\boldsymbol{\pi}$.

Because of the nature of the standing motion studied here, we may further restrict the parameterization. Since the robot starts in a seated posture and finishes in a standing posture, the initial, $\boldsymbol{\theta}_{\text{start}}$, and final, $\boldsymbol{\theta}_{\text{end}} = \boldsymbol{\theta}_{n_\theta}$, waypoints of the movement remain constant. As such, only the intermediate waypoints are used to modify $\boldsymbol{\pi}_\theta$. Here, we consider only one intermediate CoM waypoint, $\boldsymbol{\theta}_{\text{mid}}$, simplifying the policy parameterization to,

$$\boldsymbol{\pi}_\theta = \boldsymbol{\rho}(\theta_{\text{mid}}) \ . \quad (4)$$

### *D. Policy Rollouts: Task-Set Execution*

Given a parameterized policy, $\boldsymbol{\pi}_\theta$, we wish to determine the evolution of the robot's states and actions. The policy is therefore rolled-out, meaning that the task-set is executed on the robot, either in simulation or reality, and the state and action data are recorded,

$$\{\mathcal{S}, \mathcal{A}\} = \text{rollout}(\boldsymbol{\pi}_\theta) \ , \quad (5)$$

where $\mathcal{S}$ and $\mathcal{A}$ are the concatenations of the states and actions over the entire rollout. This implies that the full control architecture, as described in Sec. II-A, is employed until the task execution is complete, meaning that the execution must occur in a finite amount of time and should be finished in the duration dictated by the CoM policy $\boldsymbol{\rho}(\Theta^{\text{CoM}})$, $t^{\text{CoM}}_{\boldsymbol{\pi}}$. However, if the robot falls, then $\boldsymbol{\pi}^{\text{CoM}}$ will not be completed in $t^{\text{CoM}}_{\boldsymbol{\pi}}$. The policy rollouts are therefore assigned a maximum execution time, $t_{\max} > t^{\text{CoM}}_{\boldsymbol{\pi}}$, to allow for possible delays in the task execution but to avoid recording failed rollouts indefinitely. Here, we arbitrarily select $t_{\max} = 1.5 \times t^{\text{CoM}}_{\boldsymbol{\pi}}$.

### *E. Penalty Function: Task Feasibility Cost*

In order to evaluate the policy rollouts, we use a penalty function based on three component cost functions, which evaluate the performance of the policy and are based on generic optimal control principles. These costs are calculated a posteriori on the rollout data determined in (5). While defined with respect to the CoM task, these costs are applicable to any other form of control task.

Using the state information $\mathcal{S}$, we can determine how the CoM evolved over the course of a single rollout. We first examine how well the CoM position, $\boldsymbol{x}_{\text{CoM}}(k)$, tracked the references, $\boldsymbol{x}^{\text{ref}}_{\text{CoM}}(k)$, provided by $\boldsymbol{\pi}_\theta$, during the rollout and develop the *tracking cost*,

$$j_t = \sum_{k=0}^{N} \left\| \boldsymbol{x}_{\text{CoM}}(k) - \boldsymbol{x}^{\text{ref}}_{\text{CoM}}(k) \right\|^2 \ , \quad (6)$$

where $N$ is the total number of time steps. We define the actual total duration of the rollout, $t_{\text{end}} = N\Delta t$, where $\Delta t$ is the control sampling period, and $t_{\boldsymbol{\pi}}^{\text{CoM}} \leq t_{\text{end}} \leq t_{\max}$. If a task error is perfectly minimized by the controller, then it goes to zero, meaning that the robot perfectly executes $\boldsymbol{\pi}_\theta$. Any error in the position tracking then reflects imperfect optimization and consequently a task infeasibility associated with the current policy. We assume that the ultimate objective of the standing motion, and any point-to-point trajectory for that matter, is to reach its final waypoint. With this in mind a *goal cost* is developed,

$$j_g = \sum_{k=0}^{N} \frac{k\Delta t}{t_{\boldsymbol{\pi}}} \|\boldsymbol{x}_{\text{CoM}}(k) - \boldsymbol{\theta}_{\text{end}}\|^2 \ , \tag{7}$$

where $\boldsymbol{x}_{\text{CoM}}(k) - \boldsymbol{\theta}_{\text{end}}$ is the difference between the CoM task position at time step $k$ and the final waypoint in its trajectory. The weight of this difference increases linearly from zero with time. This means that the distance to the goal waypoint becomes more important as time elapses. Finally, we wish to determine the most energetically optimal motion, by minimizing the actions, $\boldsymbol{a}$ (i.e. the control inputs, $\boldsymbol{\tau}$) using an *energy cost*,

$$j_e = \beta \sum_{k=0}^{N} \|\boldsymbol{\tau}(k)\|^2 \ . \tag{8}$$

Energy cannot be directly compared with Cartesian distances, so the $\beta$ parameter must be introduced to scale $j_e$ for meaningful comparison with $j_t$ and $j_g$. Here, we use $\beta = 1.0\text{e}{-4}$. The penalty function, or *feasibility cost* can be calculated by summing the component costs, and averaging over $t_{\text{end}}$ to account for rollouts with different timescales,

$$j_f = \text{penalty}\left(\{\mathcal{S}, \mathcal{A}\}\right) = \frac{j_e + j_t + j_g}{t_{\text{end}}} \ . \tag{9}$$

With (9) we can estimate the feasibility of $\boldsymbol{\pi}_\theta$. However, this estimate has no absolute significance on its own. There is no threshold value for determining analytically if $\boldsymbol{\pi}_\theta$ was successful in a high-level sense (i.e. the robot stood up). Given this ambiguity, we take the $j_f^0$ of the initial $\boldsymbol{\pi}_\theta^0$ as the reference with which all other $\boldsymbol{\pi}_\theta^i$ are compared using, $\hat{j}_f^i = \frac{j_f^i}{j_f^0}$, where $i$ indicates the rollout number. This means that the initial, $\boldsymbol{\pi}_\theta^0$, has a feasibility cost equal to 1.0 and any $\boldsymbol{\pi}_\theta^i$ which produces a $\hat{j}_f^i < 1.0$ represents an improvement in task feasibility, and vice versa for $\hat{j}_f^i > 1.0$.

### F. Optimizing The Policies: Bayesian Optimization

Since the transition dynamics, $\mathcal{P}(\boldsymbol{s}(k+1)|\boldsymbol{s}(k), \boldsymbol{a}(k))$, are governed by the equations of motion with changing contacts, $\mathcal{P}$ is a discontinuous and time-varying non-linear function. Therefore, in order to optimize the policy parameters given a scalar reward or penalty, non-convex black-box solvers must be used. The downside to these solvers is that they typically require many rollouts (parameter, $\boldsymbol{\theta}_{\text{mid}}^i$, and cost, $\hat{j}_f^i$, samples) to converge on a local optimum. In humanoid robotics, rollouts are time consuming and dangerous. As a consequence, sample efficiency is of the highest importance in PS. This, in addition to the low dimensionality of the parameter space, permits the use of BO to optimize, $\boldsymbol{\theta}_{\text{mid}}$. BO derives its sample efficiency from explicitly modeling the latent parameter to cost mapping using a Gaussian Processes (GP), and then using this model, or surrogate function, to explore the parameter space. The actual minimization is performed on an acquisition function which combines the cost means and variances provided by the GP to balance exploitation with exploration [26]. In this study, the Lower Confidence Bound (LCB) acquisition function is used (see [27]) and minimized with a Covariance Matrix Adaptation Evolutionary Strategy solver (see [28]). The parameter search space is bounded using box constraints around a 3-dimensional cube of possible $\boldsymbol{\theta}_{\text{mid}}^i$, positions as shown in Fig. 3a. The incumbent solution is taken as the best parameter and cost observation from the rollouts, $\boldsymbol{\theta}_{\text{mid}}^*$ and $j_f^*$; therefore, the optimization does not depend on the sequence in which the rollouts are performed. One drawback to BO is that it does not guarantee convergence in most cases. Here, convergence is assumed when BO proposes a new $\boldsymbol{\theta}_{\text{mid}}^i$ which satisfies,

$$\left\| \boldsymbol{\theta}_{\text{mid}}^i - \boldsymbol{\theta}_{\text{mid}}^* \right\| \leq \gamma \ , \tag{10}$$

where $\gamma$ is a distance threshold, or the number of iterations has exceeded some maximum value.

**Algorithm 1** Task Feasibility Optimization

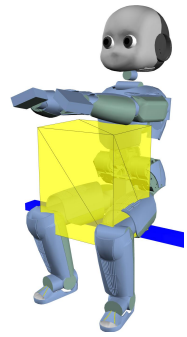
Given initial policy parameters: $\boldsymbol{\theta}_{\text{mid}}^i = \boldsymbol{\theta}_{\text{mid}}^0$.
**do**
  $\boldsymbol{\pi}_\theta^i = \boldsymbol{\rho}(\boldsymbol{\theta}_{\text{mid}}^i)$ ▷ generate policy from parameters
  $\{\mathcal{S}, \mathcal{A}\}^i = \text{rollout}(\boldsymbol{\pi}_\theta^i)$ ▷ rollout the policy
  $j_f^i = \text{penalty}\left(\{\mathcal{S}, \mathcal{A}\}^i\right)$ ▷ calculate the feasibility cost
  $\hat{j}_f^i = \frac{j_f^i}{j_f^0}$ ▷ scale the cost
  GP.Train$\left(\left\{\boldsymbol{\theta}_{\text{mid}}^i, \hat{j}_f^i\right\}\right)$ ▷ train the BO surrogate function
  $\boldsymbol{\theta}_{\text{mid}}^* = \arg\min \left\{\hat{j}_f^1, \hat{j}_f^2, \ldots \hat{j}_f^i\right\}$ ▷ get incumbent solution
  $\boldsymbol{\theta}_{\text{mid}}^i = \arg\min$ LCB ▷ minimize acquisition function
**while** (10) $\neq$ True or $i <$ Max Iter. ▷ convergence criteria
**return** $\boldsymbol{\theta}_{\text{mid}}^*$ ▷ return incumbent solution

### G. Task Feasibility Optimization

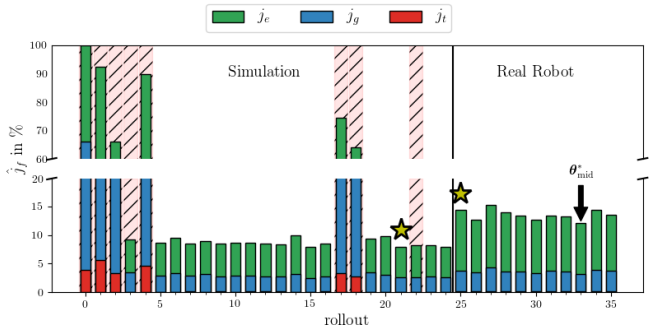
Finally, the task feasibility optimization loop can be written as shown in Algorithm 1. Starting from policy parameters $\boldsymbol{\theta}_{\text{mid}}^i = \boldsymbol{\theta}_{\text{mid}}^0$, $\boldsymbol{\pi}_\theta^i$ is generated using (4), and rolled out on either the simulated or real robot. The resulting states and actions are used to calculate a feasibility cost with (9), which is subsequently scaled. The GP of the BO surrogate function is then trained with the new parameter and cost data, $\left\{\boldsymbol{\theta}_{\text{mid}}^i, \hat{j}_f^i\right\}$, and the next $\boldsymbol{\theta}_{\text{mid}}^i$ is determined by minimizing the LCB acquisition function. The new $\boldsymbol{\theta}_{\text{mid}}^i$ is then compared to the incumbent solution $\boldsymbol{\theta}_{\text{mid}}^*$ to determine if convergence has been achieved. If so then the incumbent is returned.

## III. Experiments

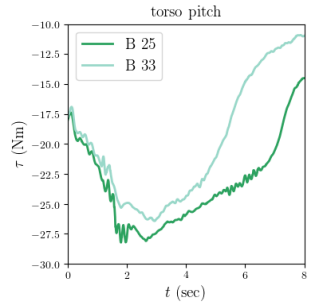
The task feasibility optimization is tested using a dynamically complex scenario in which the robot starts from a seated



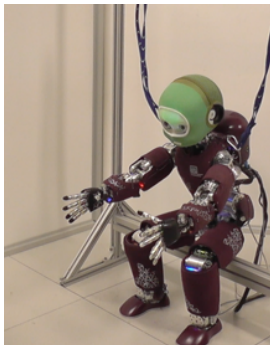
(a) $\boldsymbol{\theta}_{\text{mid}}$ bounds




(b) bootstrapped

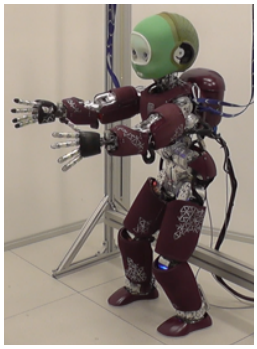



(c) torso pitch torques

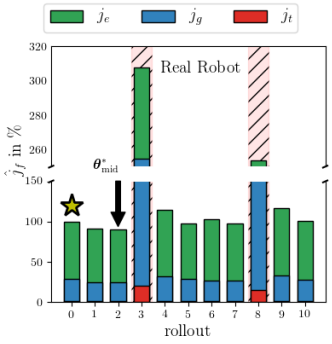


(d) 0.0s

(e) 8.0s



(f) 6.0s




(g) non-bootstrapped

Fig. 3: (a) shows the bounds initially used for the BO in simulation. For the real rollouts, these bounds are then further restricted to a 10cm cube around the initial $\boldsymbol{\theta}_{\text{mid}}$. (b) show the feasibility cost percentages (bootstrapped case) from the rollouts in both simulation and on the real robot. (c) shows the evolution of the torso pitch joint torques for the rollouts 25 and 33 in the bootstrapped case. The rollouts which produced a failure (falling) are indicated by the red hatched backgrounds. The optimal (best observed costs) policy parameters, $\boldsymbol{\theta}^*_{\text{mid}}$, are indicated for both real rollout cases. (g) shows the costs for the non-bootstrapped case. (d) shows the initial posture of the iCub robot. (e) and (f) show the final standing posture of the optimized motions for the bootstrapped and non-bootstrapped cases respectively.

position on a stationary bench and must transition to standing. The bench contacts are 22cm from the ground and on the back of the iCub's upper thigh links. The toes are in contact with the ground. The initial posture is chosen to ensure that the ground-plan ($x$-$y$) projection of starting CoM position is within the Polygon of Support (PoS) defined by the bench and ground contact locations. The contacts are managed by the FSM described in Sec. II-A. The initial policy parameters, $\boldsymbol{\theta}^0_{\text{mid}}$, are chosen between $\boldsymbol{\theta}_{\text{start}}$ and $\boldsymbol{\theta}_{\text{end}}$, resulting in a straight line CoM trajectory. A full execution of the whole-body controller constitutes a single policy rollout. The rollout is completed when the robot reaches $\boldsymbol{\theta}_{\text{end}}$ to within 3.0cm of accuracy, or if $t_{\text{end}} > t_{\text{max}}$.

The rollouts are first carried out in simulation using Gazebo as the simulation environment with the ODE physics engine. PS is iterated until one of the convergence criteria detailed in Sec. II-F is met. In this study $\gamma = 1.0$cm, and the maximum number of iterations is 30 in simulation and 10 on the real robot. The optimal policy parameters, $\boldsymbol{\theta}^*_{\text{mid}}$ are then used to generate $\boldsymbol{\pi}^*_\theta$ which is rolled out on the real iCub. This rollout is used to demonstrate that task feasibility can be initially optimized in simulation and produce feasible motions on the real robot. With the $\boldsymbol{\pi}^*_\theta$ from simulation as a starting point, the PS is continued by performing rollouts on the real iCub. For these rollouts we look at two cases. In the first, the BO surrogate function training is *bootstrapped* with training data from the simulated rollouts and further trained on data from the real rollouts. In the second *non-bootstrapped* case, the surrogate function is trained only using the real rollout data. For both cases, the $\boldsymbol{\pi}^*_\theta$ from the simulation rollouts is used as as the initial policy for the real rollouts, warm starting the PS. To limit the number of falls, the BO search space bounds are restricted to a 10cm cube around the initial $\boldsymbol{\theta}^*_{\text{mid}}$, for the real rollouts. Ten rollouts are performed for both cases. *All code and data for these experiments is open-source and can be found here: https://github.com/rlober/task-optim. Please see the accompanying video for a detailed look at the rollouts.*

## IV. Results

In Fig. 4, we see the evolution of the CoM for the original policy, B 0, and the policies optimized in simulation, B 25, the bootstrapped case, B 33, and the non-bootstrapped case, NB 2. The initial straight line CoM trajectory produces an unstable whole-body motion, which causes the robot to lose balance. The failing (i.e. falling) rollouts are indicated by the hatched red backgrounds in Figs. 3b and 3g. Because the initial policy fails, the measured CoM position values for B 0 are not shown after 2.5 seconds due to noise, and the $F_z$

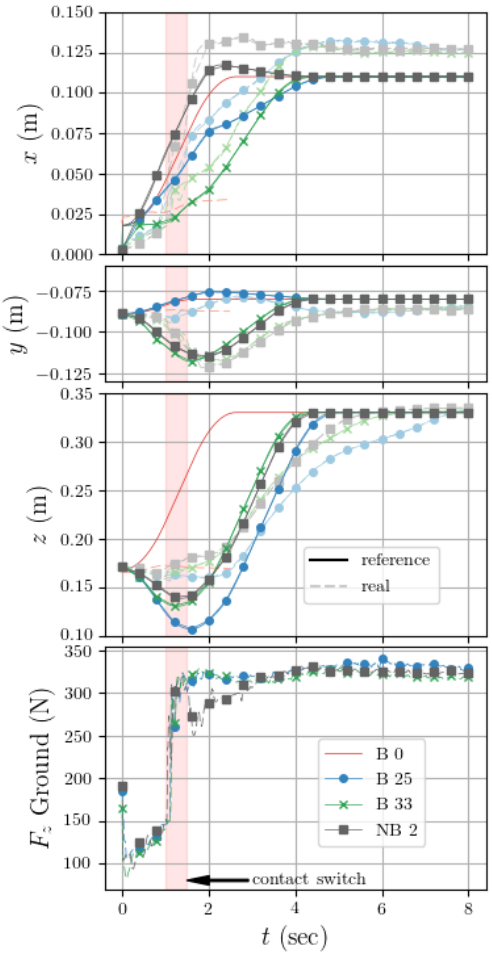



Fig. 4: The evolution of the CoM trajectories generated by the original and optimized policies. "B" indicates the bootstrapped case, and "NB" the non-bootstrapped case. B 0 is the original policy executed in simulation. The optimal policy found in the simulated rollouts comes from B 21, or the 21st rollout of the bootstrapped case. B 25 and NB 0, i.e the first real rollouts for the bootstapped and non-bootstrapped cases, use the B 21 policy. This policy is indicated by the yellow stars in the cost curves in Fig. 3b and Fig. 3g. B 33 is the optimal policy found during the real bootstrapped rollouts. NB 2 is the optimal policy found during the real non-bootstrapped rollouts. The solid lines are the reference values generated by $\boldsymbol{\pi}_\theta$ and the lighter dashed lines are the real measured values. The original, B 0, real lines are cut off after 2.5s when the robot falls. The noisy B 0 force profile is omitted from the force plot, to not obfuscate the other force profiles.

values are omitted completely for clarity. After 24 rollouts in simulation (see Fig. 3b), the task feasibility optimization converges to a policy which produces a successful sit-to-stand transition in both simulation and on the real robot. The rollouts can be watched in the accompanying video. This policy comes from the rollout 21 in simulation, and is used as the policy for the initial real rollouts in both the bootstrapped and non-bootstrapped cases, B 25 and NB 0 respectively. This is confirmed by the real and reference CoM trajectories for B 25 in Fig. 4. Had the motion failed, the real values would not have tracked the reference values as is the case for B 0.

Looking at the $z$-axis and $F_z$ plots in Fig. 4, we see that the optimal strategy, found in B 21, is to move the CoM downwards initially to increase the ground reaction force, and shift the robot's weight to the feet. This shift must come early in the execution of the CoM trajectory in order to achieve a contact switch in the FSM, and thus allow the CoM to continue tracking the trajectory references. When this policy is executed on the real robot in B 25 and NB 0, the results are successful, but higher $j_f$, than predicted by simulation, are observed for both cases. These discrepancies come as no surprise, but indicate that some unpredicted factors come into play on the real robot and must therefore be accounted for.

Looking at NB 2 and NB 3, we have an example of an optimal policy and a costly policy which produces a fall. In these two rollouts, the policy parameters being tested are $\boldsymbol{\theta}^*_{\text{mid}} = \boldsymbol{\theta}^2_{\text{mid}} = \begin{bmatrix}0.12 & -0.124 & 0.115\end{bmatrix}^\top$ and $\boldsymbol{\theta}^3_{\text{mid}} = \begin{bmatrix}0.12 & -0.02 & 0.115\end{bmatrix}^\top$, respectively. These parameters differ by only 10cm in the $y$-axis, which in theory, should not affect a sagittal plane motion. However, this subtle change in the trajectory makes the difference between optimality and catastrophic failure. We can see in the $y$-axis plot of Fig. 4 that the optimal policies found both with and without bootstrapping possess this $y$-axis motion, contrary to the policy optimized in simulation, and clearly attempt to compensate for un-modeled infeasibilities in the real system. Given the sensitive nature of the sit-to-stand motion, hand-tuning the trajectory parameters would be a difficult chore even for an expert.

Figures 3b and 3g show the component costs for each rollout with and without bootstrapping. The percentage improvement, $\hat{j}^i_f \times 100$, of each cost shows how PS improves the motion with respect to the initial policy. The overall evolution of the total feasibility costs shows the almost binary nature of the sit-to-stand scenario — either the robot stands or it falls. Given this, and the nature of the BO used here, we do not observe smooth convergence. Furthermore, in both the bootstrapped and non-bootstrapped cases the convergence criterion from (10) is not attained. Nevertheless, the initial policies are improved using task feasibility optimization. The majority of this improvement arises thanks to a decrease in energy consumption. The energy savings come primarily from the large sagittally actuated pitch joints, and most notably that of the torso pitch. In Fig. 3c, we see the torques from B 25 and B 33. Both policies produce a successful sit-to-stand motion, but the optimized policy solicits this actuator less than the initial policy and reaps large gains in the energy cost. As expected, the rollouts without bootstrapping show more aggressive exploration, with two policy failures at NB 3 and NB 8, than the rollouts with bootstrapping. This comes from the higher variance associated with the un-explored regions of the policy parameter search space. The exploration however, leads to an optimized motion which moves more quickly from the starting seated posture (see Fig. 3d) to a standing posture, as shown by the trajectory in Fig. 4, allowing it to spend less time in configurations which require

large torques, than the solution found using bootstrapping. The decreased goal costs come from the fact that the robot is already standing after only 6.0s (see Fig. 3f) rather than 8.0s as is the case with the less aggressive movement found by the bootstrapped optimization (see Fig. 3e). Around the solution space of feasible sit-to-stand CoM trajectories, the tracking cost has little impact on the total cost, but becomes more prominent when the policy fails.

## V. Conclusion

The main takeaway from this work is that by exploiting an underlying model-based control architecture, we are able to abstract the problem of producing feasible motions to only a few task-space variables, which can affect drastic changes in the overall behavior. Given the low-dimensionality of the variables, PS can be applied in a sample efficient manner, making it viable for real robots which must learn quickly and efficiently with minimal failures (e.g. humanoids). This result should not be understated because motions planned in simulation, or using approximate models, are never executed perfectly on the real robot, and the infeasibilities must be corrected or tuned in most cases. Making this correction automatic, is a crucial step towards truly autonomous robots, and cannot practically be achieved on a real system with model-based control [7] or learning [1] alone. Our generic model-free approach allows any underlying whole-body controller to be used, as shown here and in [15], and requires only the existence of task trajectories with which to optimize policies. Through the example sit-to-stand scenario, we show that task feasibility optimization provides an efficient interface between control and learning, which can resolve task infeasibilities and produce viable whole-body motions in both simulation and reality. In future work, we hope to find automated ways of determining the policy parameters which need to be optimized, rather than having to specify them by hand. An advancement such as this would render task feasibility optimization entirely self-sufficient.

## References


[1] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic," 2017.

[2] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 29–43, 2004.

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

[4] L. Saab, O. E. Ramos, F. Keith, N. Mansard, P. Soueres, and J.-Y. Fourquet, "Dynamic whole-body motion generation under rigid contacts and other unilateral constraints," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 346–362, 2013.

[5] K. Bouyarmane and A. Kheddar, "Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4414–4419.

[6] A. Ibanez, P. Bidaud, and V. Padois, "Emergence of humanoid walking behaviors from Mixed-Integer Model Predictive Control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, USA, Sept 2014, pp. 4014 – 4021.

[7] J. Koenemann, A. D. Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2015, pp. 3346–3351.

[8] K. Bouyarmane and A. Kheddar, "Humanoid robot locomotion and manipulation step planning," *Advanced Robotics*, vol. 26, no. 10, pp. 1099–1126, 2012.

[9] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, Dec 2014.

[10] K. Bouyarmane and A. Kheddar, "On Weight-Prioritized Multi-Task Control of Humanoid Robots," *IEEE Transactions on Automatic Control*, in revision 2015.

[11] P.-B. Wieber, A. Escande, D. Dimitrov, and A. Sherikov, "Geometric and numerical aspects of redundancy," in *Geometric and Numerical Foundations of Movements*, 2017.

[12] R. Lober, V. Padois, and O. Sigaud, "Variance modulated task prioritization in Whole-Body Control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2015, pp. 3944–3949.

[13] V. Modugno, G. Neumann, E. Rueckert, G. Oriolo, J. Peters, and S. Ivaldi, "Learning soft task priorities for control of redundant robots," in *IEEE International Conference on Robotics and Automation*, May 2016, pp. 221–226.

[14] F. Stulp and O. Sigaud, "Robot Skill Learning: From Reinforcement Learning to Evolution Strategies," *Paladyn Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, Aug. 2013.

[15] R. Lober, V. Padois, and O. Sigaud, "Efficient reinforcement learning for humanoid whole-body control," in *IEEE-RAS 16th International Conference on Humanoid Robots*, Nov 2016, pp. 684–689.

[16] R. Calandra, N. Gopalan, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian gait optimization for bipedal locomotion," in *International Conference on Learning and Intelligent Optimization*. Springer, 2014, pp. 274–290.

[17] R. Antonova, A. Rai, and C. G. Atkeson, "Sample efficient optimization for learning controllers for bipedal locomotion," in *IEEE-RAS International Conference on Humanoid Robots*, Nov 2016, pp. 22–28.

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

[19] P. Englert and M. Toussaint, "Combined Optimization and Reinforcement Learning for Manipulations Skills," in *Robotics: Science and Systems*, 2016.

[20] A. Dietrich, C. Ott, and A. Albu-Schäffer, "An overview of null space projections for redundant torque-controlled robots," *The International Journal of Robotics Research*, vol. 34, no. 11, pp. 1385–1400, 2015.

[21] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[22] D. Pucci, F. Romano, S. Traversaro, and F. Nori, "Highly dynamic balancing via force control," in *IEEE-RAS International Conference on Humanoid Robots*, Nov 2016, pp. 141–141.

[23] G. Nava, F. Romano, F. Nori, and D. Pucci, "Stability analysis and design of momentum-based controllers for humanoid robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2016, pp. 680–687.

[24] M. P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[25] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, 2012.

[26] E. Brochu, V. M. Cora, and N. de Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *ArXiv e-prints*, Dec. 2010.

[27] D. Cox and S. John, "A statistical method for global optimization," in *Systems, Man and Cybernetics, IEEE International Conference on*, Oct 1992, pp. 1241–1246 vol.2.

[28] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. Springer, 2006, pp. 75–102.