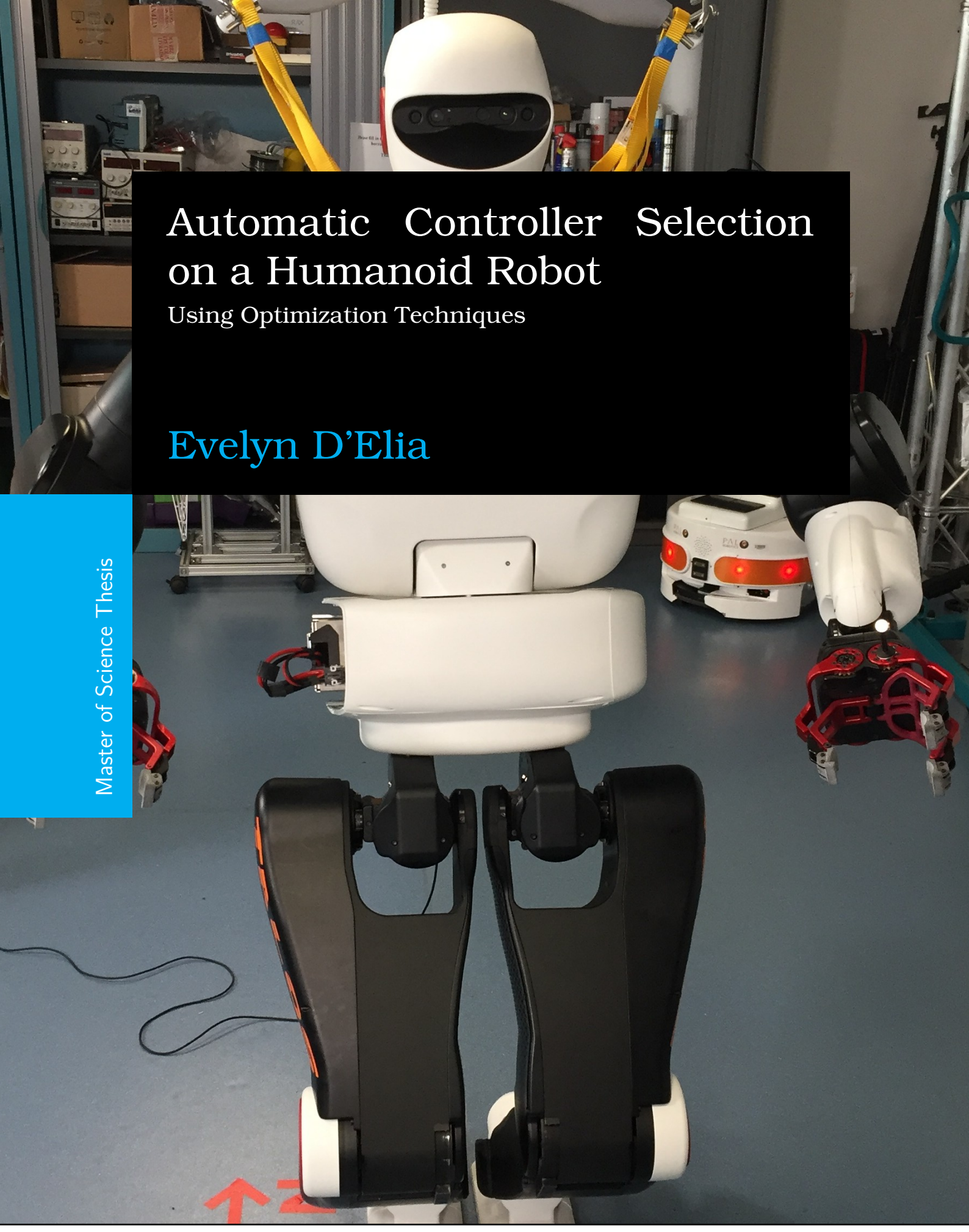


Automatic Controller Selection on a Humanoid Robot

Using Optimization Techniques

Evelyn D'Elia

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Automatic Controller Selection on a Humanoid Robot

Using Optimization Techniques

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Evelyn D'Elia

August 11, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology


The work in this thesis was conducted with the Life-long Autonomy and interaction skills for Robots in a Sensing ENvironment (LARSEN) lab at National Institute for Research in Digital Science and Technology (INRIA) Nancy, France, under the supervision of Dr.ir. Jean-Baptiste Mouret and Dr.ir. Serena Ivaldi. This work was also supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 731540 (project AnDy) and partly funded by the CPER project "CyberEntreprises" and the CPER project SCIARAT.


Abstract

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

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors at INRIA, dr.ir. Jean-Baptiste Mouret and dr.ir. Serena Ivaldi, for always making time to discuss my project and for providing valuable advice and guidance over the past six months. I would also like to thank my advisor, prof.dr.ir. Jens Kober, for providing thoughtful feedback and comments on my work throughout the evolution of my thesis, and prof.dr.ir. Luca Laurenti for serving on the evaluation committee.

In addition, I would be remiss not to acknowledge the help and support of my colleagues, with whom I exchanged ideas, tips, and tricks on a wide range of topics. Thanks to Waldez for the presentation advice and wise words. Thanks also to Ivan and Eloise, without whom testing on the real robot would not have been possible. Another thank you to Edoardo, for providing suggestions and encouragement during long hours spent debugging. I would like to thank everyone in the Larsen lab for making my time there inspiring and enjoyable.

Finally, I want to acknowledge and thank Gennaro and my parents for being my biggest fans throughout this entire process. Without them, completing my thesis in this confusing pandemic period would have been all the more difficult.

Delft, University of Technology — Evelyn D'Elia
August 11, 2021

Chapter 1

Introduction

Humanity's dream of human-like machines is as old as civilization itself. Automatons were even present in Greek mythology, thousands of years ago: the ancient Greeks believed that the god Hepaestus built a bronze automaton named Talos [1]. So, it is not by coincidence that the robot used in this work bears the same name, Talos [2]. Then and now, humans have desired to create robots that would do their work for them. 2000 years later, we now have the ability to achieve this dream.

1-1 Humanoid robots

Although technology is much more advanced than it was during the time of the Greek empire, researchers still encounter a myriad of difficulties when controlling humanoid robots. Despite our best efforts, we still often find our robots in a similar configuration as shown in Figure 1-1. Nonetheless, humans are stubborn and we continue to persist in developing better ways to control humanoids.

There are a variety of motivating factors for developing robust, high-performing humanoid robots. A robot that is shaped and moves like a person can work in a workspace designed for humans without the need to tailor the environment. Ideally, a humanoid robot should be able to complete manual labor that is grueling or even unsafe for humans. However, replicating the mechanics of the human body on a machine requires a high number of degrees of freedom (DoF), and any controller for a robot with such a large state and action space will have high computational complexity.

Because of this complexity, and because of the ambition to make the robot capable of performing vastly different trajectories such as locomotion or manipulation, it is difficult to find successful controllers. One popular way of solving this problem is by breaking the control down into a set of tasks, each with certain gain and weight parameters to define their behavior. This is called a task priority-based approach, as explained in [4], and it unifies the control of separate parts of the robot into a single, whole-body control framework. This makes controller design much more straightforward, but it still requires the task parameters to be

Figure 1-1: Fallen robot during the 2015 DARPA Robotics Challenge [3].

chosen. One typical way to choose them is by hand-tuning these parameters, but since this method is time-consuming, it often focuses on tuning a single controller which works for a large set of trajectories. The main drawback of this method is that with a single "robust" controller, the quality of individual trajectories is compromised.

In addition to the problem of designing a stable, accurate controller for a humanoid robot, there is also the problem of creating control parameter sets that are transferable. Transferability in robotics is the ability to adapt to new situations, namely new trajectories or a real robot which behaves differently from the simulation model. This thesis aims to address both of these problems in two steps: a multi-objective optimization (MOO) step to generate a large set of control parameter options, and a Bayesian optimization (BO) step to narrow down these solutions to one which works for a new, untrained trajectory or a new robot model.

1-2 Multi-objective optimization

In most cases of learning on a real robot, especially a complex humanoid, it is beneficial to first learn controllers in simulation. One common means of doing this is via mathematical optimization [5], where variables in an objective function are tested, changed, and iterated until the function value is minimized or maximized. Several types of optimization methods have been applied to the robot control problem, the most relevant of which is multi-objective optimization. Multi-objective algorithms are of interest because robot performance is often measured in many ways at once. This type of method requires a prohibitive number of trials, which constrains it to simulation use, but it can be employed as a jumping-off point for other, more efficient real-world learning algorithms. One way MOO has been used in previous work is to control a humanoid robot to show that creating a set of simulation-trained Pareto-optimal controllers lowers the number of tests that must be done on the real robot [6]. However, since this method trades off only two objectives, accuracy and stability, averaged out over the training trajectories, it does not take advantage of the full capabilities of the MOO algorithm used.

To ensure that there are highly accurate controllers for different types of trajectories, and to save time spent hand-tuning, in this thesis the use of NSGA-II [7], a multi-objective optimization algorithm, is proposed to learn a Pareto front of task parameter sets for the Talos robot that are scored based on how they perform for each of a set of training trajectories. This approach directly optimizes based on the performance of a parameter set on each trajectory, which produces a more diverse Pareto front with a better chance of transferring to new trajectories.

1-3 Bayesian optimization

Once MOO has been used to generate a Pareto front of possible control solutions, this thesis also introduces the idea of using that Pareto front as the search space for a single-objective data-efficient learning algorithm. This can specifically be used to transfer Pareto solutions to new trajectories that are not part of the MOO training set, and choosing a data-efficient algorithm leaves open the possibility of applying the approach on the real robot.

One data-efficient class of learning algorithms, called policy search (PS) [8], [9], is very promising in the context of real-world robot learning as a method that does the opposite of deep learning: its goal is to learn a successful controller with as little data as possible. The reason PS is useful for data efficiency is because it allows the dimensionality of the problem to be shrunk down to the number of parameters in the policy. Sometimes, learning a model of the dynamics or the expected return concurrently with learning the policy parameters can further decrease the number of needed trials. The most successful PS methods also aim to represent the uncertainty of the models as a way of avoiding overfitting to the few trials being conducted and increasing the robustness of the learned controller. PS can be carried out in even fewer trials if it is fed prior information in the form of a Pareto front to search.

In terms of learning with very few trials on the real robot, the success of BO [10], a PS method that models the expected return as a stochastic process, surpasses most other PS algorithms. The model of mean and variance of the expected return allows the learned controller to be robust despite few trials. One of the few downsides of BO is that it does not scale well to high-dimensional parameter spaces, but since the size of the Pareto solutions generated in this thesis work is relatively small at 10-D, it is a perfect choice.

1-4 Contributions

To summarize, the proposed contributions of this thesis are as follows:

- use the Non-dominated Sorting Genetic Algorithm II (NSGA-II) MOO algorithm with a separate objective function for each training trajectory in order to generate a diverse Pareto front of possible task parameter sets for the robot,
- compare Pareto front solutions to hand-tuned solutions on the real Talos, and
- implement a BO algorithm which searches along this Pareto front to quickly find suitable parameter sets for new robot trajectories.

The results of this analysis surprisingly show that between different simulated robot models, the Pareto fronts generated by NSGA-II actually perform very similarly, which indicates that there may be no need to use BO for closing the reality gap between simulation and physical robot. This is why we focus instead on achieving transfer to novel trajectories. The first item of contribution of this work was also presented as a poster at the 2021 Legged Robots ICRA Workshop. A video of this presentation can be found at https://youtu.be/TaZooEwb3SE.

1-5 Outline

This thesis is structured as follows:

- Chapter 2 discusses the necessary background information surrounding the thesis topic. It first explains the details of the robot and the control framework used. Then, it addresses the concept of Pareto optimality, some related work on MOO for robotic control, and the MOO algorithm that is used in this project. Next, PS methods, especially BO, are discussed. The parts of the BO algorithm, its design options, and recent related work are also mentioned. The chapter concludes by tying the parts of the project together and asserting the contributions of this work.
- Chapter 3 describes the full process of optimizing task parameters with NSGA-II. At the beginning, the methods of task and objective function formulation are mentioned. Next, the specific tasks, objective functions, training trajectories, modified robot models, and other experimental design details are described. Then, results and analysis for optimization with and without self-collision checking are revealed and compared. After that the construction of the experiments on the real Talos robot are explained and the results are discussed. Finally, the implications of the results are reviewed.
- Chapter 4 deals with the second part of this thesis, BO. It begins by addressing the methods used to construct this implementation of the algorithm. The chapter continues with the testing choices, then dissects the results of both continuous-space BO, and BO along a pre-computed Pareto front. The chapter then compares the two BO approaches, and ends with a discussion on the efficacy of the methods.
- Chapter 5, the final chapter, considers the quality and importance of the results of this thesis, and then concludes with a discussion of recommended future work on the topic.

Chapter 2

Background

This thesis aims to address two main challenges associated with learning control parameters for humanoid robots: making the learning process efficient and transferring learned solutions to new situations. In order to discuss this subject matter it is first necessary to explain the building blocks of the approaches used and and related work on the topics. This chapter provides background information about the robot, the control framework used, multi-objective optimization (MOO), and Bayesian optimization (BO). It also discusses and evaluates the efficacy of other approaches similar to the one in this project.

2-1 Talos robot

The robot used in both simulation and real-world experiments for this thesis is the Talos humanoid from PAL Robotics. The robot, shown in Figure 2-1, is roughly the size of an adult human and has 32 degrees of freedom (DoF) [2]. It is designed specifically to be capable of lifting heavy objects and has been shown to be able to hold 6 kg weights in each hand with its arms extended out to the side.

In order to control a mobile legged robot like this one, it is essential to prevent the robot from falling over. For this reason, the concept of a support polygon (SP) is used: a support polygon is defined as the convex polygon which is bounded at the outer edges of the robot's feet. In order for the robot not to fall, its zero moment point (ZMP), i.e. the 2-D ground projection of the center of mass (CoM), must stay within the SP at all times during movement [11]. These concepts of robot stability are illustrated in Figure 2-2.

2-2 Task priority-based control

The control framework employed on the Talos is a common type of whole-body control (WBC) formulation known as task priority-based control. Here a task is defined, depending on its type, as either a Cartesian position and/or orientation, or a set of joint angles, to track.

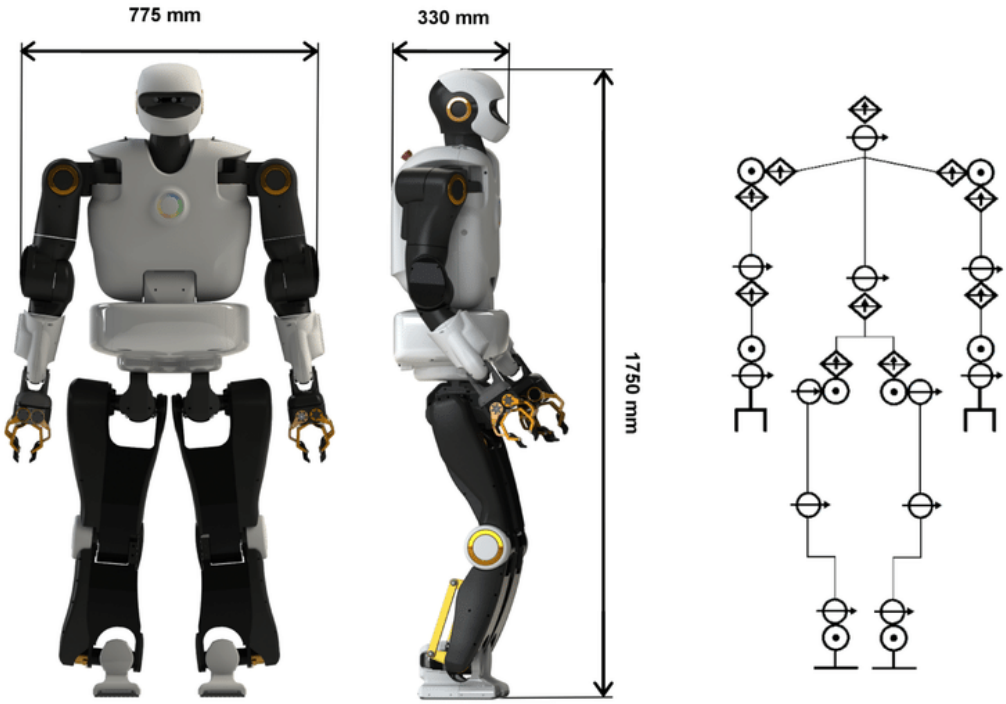


Figure 2-1: Talos humanoid robot and joint configuration [2].

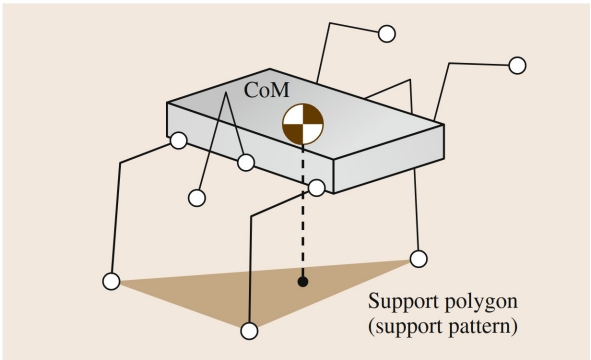


Figure 2-2: Graphic of SP and zero-moment point (ZMP, ground projection of CoM) of a legged robot [11, Figure 17.37].

Generally the assigned trajectory for the robot to follow is broken down with a task priority-based framework into pieces to track such as the CoM Cartesian position or foot orientation.

Hard and soft task priorities are defined as explained in [4]. Soft priorities are represented by a soft priority weight (SPW) w_i (i is the task index) on the error of each task $\mathcal{T}_i$, while hard priorities are represented by the hierarchy level selector l_i and executed by solving each level in the nullspace of the one above it, using the successive nullspace projection [12]. For a formulation in which both hard and soft task priorities are used, the following quadratic programming (QP) optimization problem is solved at each time step for any given level l_i of the task hierarchy:

$$\begin{aligned} \min_{u} \quad & \sum_i w_i \|\boldsymbol{A}_i \boldsymbol{u} - \boldsymbol{b}_i\|^2 + \epsilon \|\boldsymbol{u}\|^2 \\ \text{s.t.} \quad & \boldsymbol{c}_{min} \leq \boldsymbol{C}\boldsymbol{u} \leq \boldsymbol{c}_{max} \\ & \boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}, \end{aligned} \tag{2-1}$$

where $\boldsymbol{A}_i$ is the task's equivalent Jacobian, $\boldsymbol{u}$ is the control input, $\boldsymbol{b}_i$ is the reference, and ϵ is a regularization factor. The constraint inequalities represent all control input, dynamics, environmental, and other constraints on the robot. The reference $\boldsymbol{b}_i$, which depends on the desired trajectory, is defined for both Cartesian and postural tasks as:

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

where superscript d refers to desired value, $\ddot{\boldsymbol{p}}_i^d$, $\dot{\boldsymbol{A}}_i$ and $\boldsymbol{e} = \boldsymbol{p}_i - \boldsymbol{p}_i^d$, $\dot{\boldsymbol{e}} = \dot{\boldsymbol{p}}_i - \dot{\boldsymbol{p}}_i^d$ are the errors between desired and actual Cartesian position $\boldsymbol{p}_i$ and velocity $\dot{\boldsymbol{p}}_i$, and λ_i^P and λ_i^D are the proportional and derivative task convergence gains (CGs), according to [4]. Once the control input values are determined by Equation 2-1, the following equation is used to estimate the robot's response, in terms of joint positions and velocities, to those inputs:

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}), \tag{2-3}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the joint position, velocity, and acceleration vectors respectively, $\boldsymbol{M}(\boldsymbol{q})$ is the robot's mass and inertia matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is its Coriolis matrix, and $\boldsymbol{g}(\boldsymbol{q})$ is its gravitational vector. The resulting $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ from Equation 2-3 are then used as feedback to determine the next $\boldsymbol{b}_i$ in Equation 2-2, which is in turn used in Equation 2-1, and the cycle is repeated at each time step until the end of the simulation.

Task priority-based control boils down the tunable controller parameters to a small set. This set, however, nonetheless needs to be tuned, and hand-tuning is time-consuming. Rather than hand-tuning the parameters, it is possible to learn them with an optimization algorithm.

2-3 Evolutionary algorithms in robotics

The most common approach to optimization is via gradient-based methods, for which the gradient of a cost function is computed. However, this requires the objective function to be differentiable, and in this thesis that is not the case, because the objective function is discontinuous. When gradient-based optimization is impossible, a viable alternative is an evolutionary algorithm, inspired by biological evolution: in each generation, sets of parent solutions are chosen from a population and offspring solutions are created (mating selection),

the offspring solutions undergo changes (mutation), and the population size is reduced (natural selection) [13]. Evolutionary algorithms are black box optimization algorithms that work well in continuous space, which makes them useful for robotics.

One well-known and often used example of an evolutionary stochastic optimization method is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14]. An iteration in basic CMA-ES consists of taking many parameter set samples from a Gaussian distribution. Each candidate parameter solution is tested on the objective function $J(\boldsymbol{\theta})$ and the best-performing candidates are used to choose how to update the Gaussian distribution for the next iteration. However, one feature that the original version of CMA-ES lacks is the ability to apply constraints; this is solved by modified versions of the algorithm, of which (1+1) - CMA-ES with Constrained Covariance Adaptation (CCA) has been found in [15] to perform the best for a humanoid robot. In this variant of the algorithm, only one parameter set is taken in each iteration and the optimization constraints are enforced by updating the covariance to discourage solutions outside the feasible region [16]. [17] and [18] both employ this CMA-ES version to search for optimal policy parameters for simulated tasks on the child-sized humanoid iCub [19].

In [17], (1+1) - CMA-ES with CCA is used to optimize the parameters of radial basis functions (RBFs) that represent task trajectories. An unconstrained optimization step (without CCA) is done first to find an optimum within the feasible region, then that optimum is bootstrapped by a constrained version of the same stochastic optimization. The method is evaluated only in simulation on the humanoid iCub, for a single task in which the robot must stand up from a chair. A similar approach [18] represents task trajectories as probabilistic movement primitives (ProMPs), where the policy parameters optimized are one weight for each basis function in the ProMP for each task (in this case CoM and waist trajectories). The authors of [18] skip the first step in [17] by instead using retargeted recorded human trajectories to initialize the optimization problem, which has 10 parameters, and constrains the CoM to stay above ground and the robot's waist to stay close to an inertial reference. However, this study highlights an important weakness of this optimization algorithm: it must be initialized with a set of parameters within the feasible region, otherwise it cannot find a solution. Another limitation of (1+1) - CMA-ES with CCA is that it is a single objective optimization algorithm [15], [18] and thus can only train on a single trajectory per optimization run. Therefore, for learning many trajectories at once, CMA-ES with CCA is not suitable.

2-4 Multi-objective optimization algorithms

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

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \{f_1(\boldsymbol{\theta}), f_2(\boldsymbol{\theta}), \ldots, f_l(\boldsymbol{\theta})\} \\ \text{s.t.} \quad & \boldsymbol{\theta} \in S, \end{aligned} \tag{2-4}$$

where $f(\boldsymbol{\theta})$ is an objective function, l is the number of objective functions and S is the feasible set to which the parameter vector $\boldsymbol{\theta}$ must belong (i.e., it represents the optimization constraints). In problems with more than one objective function, there seldom exists a single solution that optimizes every objective function. Instead, the algorithm optimizes many

objective functions at once, without aggregating them together in a single metric. Here, the concept of Pareto optimality is often used: a Pareto optimal solution is a point in parameter space (within the feasible set S) for which no one objective function can be further optimized without compromising the performance of another objective [20]. Solving a MOO problem results in a set of Pareto optimal solutions, called the Pareto front, which each perform differently with respect to each of the various objective functions. Figure 2-3 provides a visual example of the need for multi-objective optimization and the typical appearance of a Pareto front. The example shows a situation where there are two objective functions that cannot both be optimized at once because their minima are different. Instead of finding a single optimal solution, a 2-D Pareto front can be made, where each point on the front represents the performance of a parameter set that minimizes one of the two functions without sacrificing the other.

One of the most popular MOO methods is called Non-dominated Sorting Genetic Algorithm II (NSGA-II) [7]. NSGA-II is part of a class of evolutionary algorithms, whose strength lies in their ability to find a diverse set of Pareto optimal solutions, an important characteristic for finding a robot controller that is able to generalize to many types of tasks. NSGA-II in particular is shown to outperform other evolutionary MOO methods in terms of the diversity and accuracy of solutions found [7].

NSGA-II is a genetic algorithm which works by, in each generation of the algorithm, creating a population of parameter sets $\boldsymbol{\theta}$ to test, evaluating each of them, and selecting only the Pareto solutions from that population to survive to the next generation. In the next generation, new variations and the previously created Pareto solutions are added to the population and the iterations continue until the specified maximum number of generations is reached.

2-5 Related work with NSGA-II

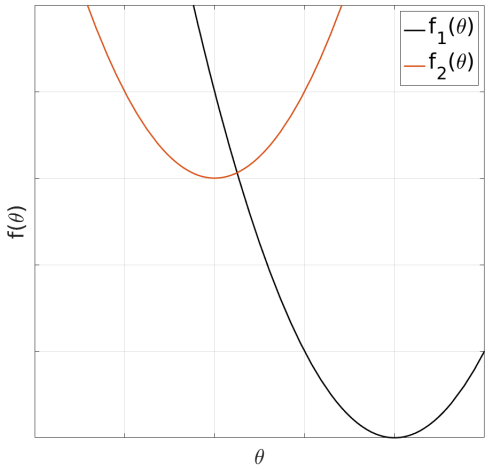
The work in this thesis builds off a similar approach by Penco et. al. [6], which also employs MOO on a humanoid robot to learn task priority weights and gains. In [6], NSGA-II is used to produce a Pareto front of control parameters for the humanoid iCub robot. In this case, the control parameters include the proportional and derivative convergence gains as well as the values that describe the hard and soft task priorities. The authors of [6], rather than defining the reference task with Equation 2-2, define the reference $\boldsymbol{b}_i$ for a Cartesian task as

$$\boldsymbol{b}_i = \begin{bmatrix} \boldsymbol{b}_{p,i}^\top & \boldsymbol{b}_{o,i}^\top \end{bmatrix}^\top = \begin{bmatrix} \left(\dot{\boldsymbol{p}}_i^d + \lambda_i \boldsymbol{e}_p\right)^\top & \left(\boldsymbol{\omega}_i^d + \sigma_i \boldsymbol{e}_o\right)^\top \end{bmatrix}^\top, \tag{2-5}$$

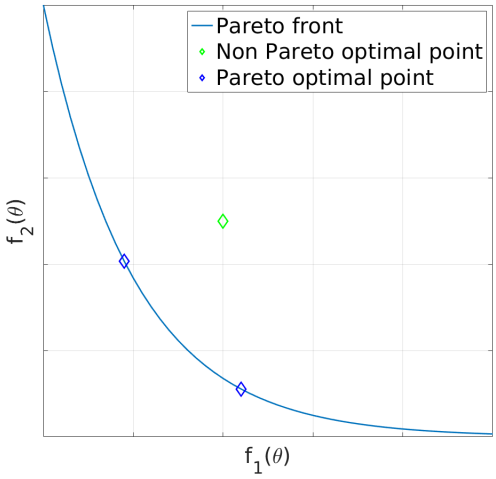
and for a postural task as

$$\boldsymbol{b}_i = \dot{\boldsymbol{q}}_i^d + \mu_i \boldsymbol{e}, \tag{2-6}$$

where i is the task index, subscripts p and o are for position and orientation in the Cartesian task, $\dot{\boldsymbol{p}}_i^d$, $\dot{\boldsymbol{\omega}}_i^d$, and $\dot{\boldsymbol{q}}_i^d$ are the desired Cartesian linear velocity, Cartesian angular velocity, and joint velocity respectively, and $\boldsymbol{e}= \boldsymbol{p}_i - \boldsymbol{p}_i^d$ is the error between desired and actual position, according to [4]. The convergence gains to be optimized are one or both of λ_i and σ_i for each Cartesian position task, or μ_i for each postural task.



(a) Plot of two functions with different minima, which cannot both be minimized with a single parameter set $\boldsymbol{\theta}$.



(b) Plot of a Pareto front and non-Pareto point. A point is non-Pareto optimal if for any objective the performance can be further improved without sacrificing the performance of another objective.

Figure 2-3: Visual aid to show the need for MOO and the typical appearance of a 2-D Pareto front.

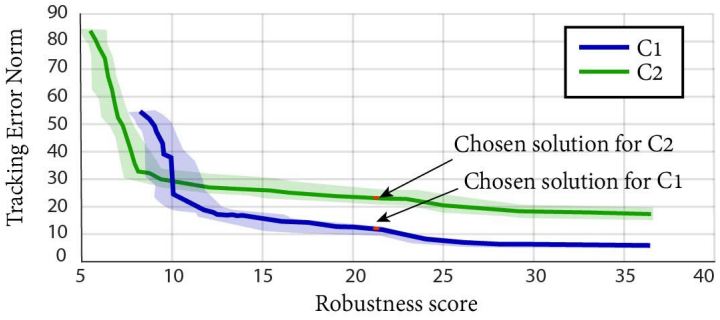


Figure 2-4: The 2-D Pareto parameter sets learned in [6, Figure 3]. Points on the upper left side of this plot are very stable but do not closely track the desired trajectory, while points on the lower right track the trajectory well but will probably cause the physical robot to fall over.

Two objective functions are employed in [6]. One measures accuracy by summing reference error over the whole rollout for each task considered:

$$f_a(\boldsymbol{\theta}) = \sum_{t,i} \left| \Phi^d_{t,i} - \Phi_{t,i} \right|, \tag{2-7}$$

where Φ is the position, orientation, and/or postural value for a given task (the function parameters), t is the time step, and superscript d indicates the desired value. The other function measures a solution's stability in terms of the distance of the ZMP from the center of the SP:

$$f_{r,t}(\boldsymbol{\theta}) = f_{r,t-1} + \begin{cases} |x^{cz}_t| + |y^{cz}_t|, & \text{if robot not fallen} \\ x_{SP} + y_{SP}, & \text{if robot fallen,} \end{cases} \tag{2-8}$$

where x^{cz}, y^{cz} are the function parameters representing the frontal and horizontal ZMP distance from the center of the SP, and x_{SP} and y_{SP} are the dimensions of the SP.

In all, their approach optimizes three to four parameters per task considered in a rollout: the soft priority weight, hierarchy level selector, and the convergence gain(s). NSGA-II uses three fitness functions to evaluate the parameters: Equations 2-7 and 2-8, and a third, 2-9, which checks whether the robot has fallen. This third function is formulated as

$$f_{f,t}(\boldsymbol{\theta}) = f_{f,t-1} + \begin{cases} 0, & \text{if robot not fallen} \\ -\alpha, & \text{if robot fallen,} \end{cases} \tag{2-9}$$

where α is a penalty for falling.

The optimization is carried out on a simulated version of the humanoid iCub robot and outputs 20 Pareto-optimal controller configurations. The resultant Pareto fronts for two different controller types are shown in Figure 2-4. After completing MOO in simulation, the authors were able to achieve reasonably robust execution of three different double-support motions on the real robot. Their method of using MOO narrows down the number of controllers to test in the real world, but it still requires a human operator to test the Pareto solutions on the real robot and choose one. Also, due to the relatively small number of objective functions (two) and training trajectories (three), the resultant Pareto front is less diverse.

2-6 Policy search methods

Policy search (PS) is a specific type of approach that falls under the heading of reinforcement learning (RL), in which the agent (decision-making controller) searches for the policy, or state-to-action mapping, that results in the optimization of an objective function. In RL, an agent uses an action $\boldsymbol{u}_t$ to interact with its environment, and then chooses the next action based on the consequent state $\boldsymbol{x}_t$ and immediate reward r_t observed from the environment (see [21, Fig. 3.1] for a diagram). The sum of rewards over time is called the return. In robotics, the action is often expressed in terms of control inputs such as a vector of joint angles, while the state is often approximated by the information obtained from the robot's sensors, such as cameras and accelerometers. One way in which an RL method can be categorized is as either a value function-based approach or a policy search (PS) approach [22]. A value function-based approach is done in two steps: first it estimates the value function, then maximizes it to determine an optimal policy. This type of approach has the potential to directly find an optimal solution, but tends to be more complex to solve. Instead, PS directly optimizes the policy by maximizing expected return. Since the two-step value-based approach permits more error propagation and often increases the dimensionality of the problem, PS is a better choice for robotics, where the tasks are episodic, state, action, and parameter spaces can be large or continuous, and the search method is more careful with less opportunity for errors to grow [22].

Recently in the RL field there has been a focus on big data and deep learning, but most of these algorithms focus on using as much data as possible to get an accurate result, and thus are feasible only in simulation. For a real robot, this is not the ideal approach because it is difficult and inefficient to run a large number of trials. In contrast, PS is very promising for real rollouts because its framework facilitates training an agent using only a small amount of data.

There are three main types of PS: model-free (has no system model), model-based (uses a model of the system and uses it to estimate expected return), and surrogate-based (uses a model of the expected return). The surrogate-based approach makes the parameter search more efficient by learning a model of the expected return (objective function $\hat{J}(\boldsymbol{\theta})$). This type of method has been shown to outperform the other methods in terms of minimizing the number of trials required [9].

2-7 Bayesian optimization

The second part of this thesis project aims to facilitate transfer of Pareto solutions from the MOO step to the real robot. For this, a surrogate-based PS method [9] is chosen because this type of algorithm conforms to this particular problem's need for few rollouts. The specific method used, BO, is a useful method for robots like humanoids whose dynamics are complex, because instead of directly representing the expected return, BO employs a parametrized model of it that can be learned, limiting the influence of model inaccuracies by making the model probabilistic.

After each rollout, BO learns a model of the expected return. The surrogate model that is

almost always used is a Gaussian process (GP):

$$\hat{J}(\boldsymbol{\theta}) \sim \mathcal{GP}(\mu(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}')), \tag{2-10}$$

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

$$\begin{aligned} \mu(\boldsymbol{\theta}_*) &= \boldsymbol{k}^\top K^{-1} D_{1:t} \\ \sigma^2(\boldsymbol{\theta}_*) &= k(\boldsymbol{\theta}_*, \boldsymbol{\theta}_*) - \boldsymbol{k}^\top K^{-1} \boldsymbol{k}, \end{aligned} \tag{2-11}$$

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

The kernel function $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, used to quantify the difference in expected return $\hat{J}(\boldsymbol{\theta})$ between two possible parameter sets, can be defined in various ways to shape the performance of the optimization. A class of commonly used kernels, called Matérn kernels, so classified because of their form, are useful because of their flexibility [10]. The most popular Matérn kernel is the Squared Exponential (SE) distance kernel, which essentially depends on Euclidean distance between the two inputs:

$$k_{SE}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sigma_k^2 \exp\left\{-\frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)^\top \mathrm{diag}(\boldsymbol{l}_k^2)(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)\right\}, \tag{2-12}$$

where σ_k^2 and $\boldsymbol{l}_k$ are the signal variance and length scale vector hyperparameters respectively. Another such kernel, the $\frac{5}{2}$ Matérn kernel, is the one used for the implementation in this project:

$$\begin{aligned} k(\boldsymbol{\theta}, \boldsymbol{\theta}') &= \sigma_k^2 \left(1 + \frac{\sqrt{5}r}{l_k} + \frac{5r^2}{3l_k^2}\right) \exp\left(-\frac{\sqrt{5}r}{l_k}\right) \\ r &= \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|, \end{aligned} \tag{2-13}$$

In BO, the acquisition function uses the surrogate model $\hat{J}(\boldsymbol{\theta})$ to choose which policy parameters to test next, while balancing exploration and exploitation. Typically in order to find parameter sets to look at, a nonlinear optimizer is used. An acquisition function is defined as the expected utility of a given parameter vector $\boldsymbol{\theta}$, where the utility is a measure of how much information will be provided by a parameter set. Figure 2-5 shows, for a 1-D example, how the location of the maximum of the acquisition function determines the next sampled parameter $\boldsymbol{\theta}$.

Some of the most common acquisition functions are probability of improvement (PI) [23], expected improvement (EI) [24], and Upper Confidence Bound (UCB) [25]. Both PI and EI check the likelihood that a given parameter vector will improve the return beyond a target value T, which must be tuned. Of these, the most promising acquisition function, UCB [25], is optimistic in that it judges each prospective parameter set by the upper limit of the GP of its expected return:

$$\alpha_{UCB}(\boldsymbol{\theta}; D_{1:t}) = \mu(\boldsymbol{\theta}) + \beta\sigma(\boldsymbol{\theta}), \tag{2-14}$$

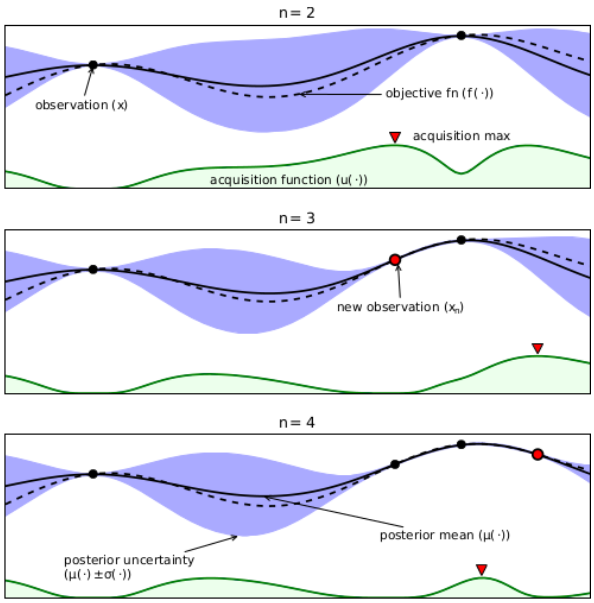


Figure 2-5: 1-D example of BO, showing how each sampled parameter set is picked by the acquisition function [10].

where β is a hyperparameter that balances exploitation and exploration by specifying how high the upper confidence is. This hyperparameter can be tuned by hand according to the needs of the application, or can instead be calculated, in a variant of the UCB algorithm known as GP-UCB. An analysis of three different acquisition functions including UCB shows that UCB is clearly advantageous for making sure there is enough exploration of the parameter space [26]. In the GP variant of UCB, β is calculated as:

$$\beta = \sqrt{2 \log \left(\frac{n^{\frac{d}{2}+2} \pi^2}{3\delta} \right)}, \tag{2-15}$$

where n is the number of past data points, δ is a hyperparameter between 0 and 1, and d is the dimension of the parameter vector.

Figure 2-6 shows the steps of the BO algorithm: first perform a rollout to measure the fitness of a solution, then update the GP to reflect the results of that rollout, and finally evaluate the acquisition function to choose the next parameter set $\boldsymbol{\theta}$ to test.

2-8 Related work with BO

BO is, in terms of data efficiency, the best PS approach for this work. It can be tailored to fit the needs of the user by picking and choosing its elements, such as the mean function $\mu(\boldsymbol{\theta})$, the covariance function $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, and the acquisition function $\alpha(\boldsymbol{\theta}; D_{1:t})$. Table 2-1 summarizes the most relevant recent work in BO.

Algorithm 3: Policy Search With Bayesian Optimization.

1: **procedure** COLLECTSTRATEGY
2: Collect samples of the form $(\boldsymbol{\theta}, R(\boldsymbol{\tau}))$
3: **end procedure**
4: **procedure** MODELSTRATEGY
5: Learn model $\hat{J} : \boldsymbol{\theta} \rightarrow J(\boldsymbol{\theta})$
6: **end procedure**
7: **procedure** UPDATESTRATEGY
8: $\boldsymbol{\theta}_{n+1} = \arg\max_{\boldsymbol{\theta}}$ ACQUISITIONFUNCTION$(\boldsymbol{\theta}|\hat{J})$
9: **end procedure**

Figure 2-6: Algorithm outline of BO [9].

Pub	Algorithm	Robot	Task	Mode	Param space	State space	Joint space	Trials
[27]	BO w/UCB	biped	walk	real	$\mathbb{R}^4$	$\mathbb{R}^4$	$\mathbb{R}^4$	~25
[28]	BO w/ DoG kernel	planar biped	walk	simulation	$\mathbb{R}^{16}$	$\mathbb{R}^{44}$	$\mathbb{R}^{14}$	~25
[29]	BO w/ trajNN kernel	planar biped	walk	simulation	$\mathbb{R}^{16}$	$\mathbb{R}^{44}$	$\mathbb{R}^{14}$	~20
[30]	BO w/ UCB	damaged hexapod	walk	simulation	$\mathbb{R}^{36}$	$\mathbb{R}^3$	$\mathbb{R}^{12}$	~10

Table 2-1: Summary of recent BO approaches on legged robots and the experiments used to validate them.

2-8-1 Incorporating prior information

Prior information has the potential to speed up learning drastically. In BO, the typical way of providing this prior information is via the GP mean and covariance functions.

One successful use of priors in BO [30] pre-computes a behavior-performance map, or a map in parameter space of how well the robot performs in simulation, before commencing the optimization. This map becomes the initial guess for the GP mean function $\mu(\boldsymbol{\theta})$ and thus serves as a guide for learning on the real robot, saving time by guiding the real-world BO toward likely high-performing solutions.

Other works incorporate priors into the kernel function instead. Some of these are tailored to the type of robot or task, and some of them take advantage of the flexibility of a neural network (NN). For example, the behavior-based kernel (BBK) introduced in [31] compares the policy distributions by calculating the Kullback Leibler (KL) divergence, but it is done in a very computationally costly way. In [28] a new type of kernel, called Determinants of Gait (DoG), was designed to implement BO on a bipedal robot. This kernel is designed specifically to teach a bipedal robot with 16 policy parameters to walk, and uses short simulations to evaluate the success of a gait by checking whether the torso is leaning forward and whether the knee is bent while the leg is in the air, among other criteria. Using simulations to construct the kernel is a way of incorporating prior information to speed up the PS. After the short simulation, a composite function $\phi_{DoG}(\boldsymbol{\theta})$ that accounts for the aforementioned performance measures is then used to construct the kernel k_{DoG}:

$$k_{DoG}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sigma_k^2 \exp\left\{-\frac{1}{2}(\phi_{DoG}(\boldsymbol{\theta}_i) - \phi_{DoG}(\boldsymbol{\theta}_j))^\top \mathrm{diag}(\boldsymbol{l}_k^2)(\phi_{DoG}(\boldsymbol{\theta}_i) - \phi_{DoG}(\boldsymbol{\theta}_j))\right\}, \quad (2\text{-}16)$$

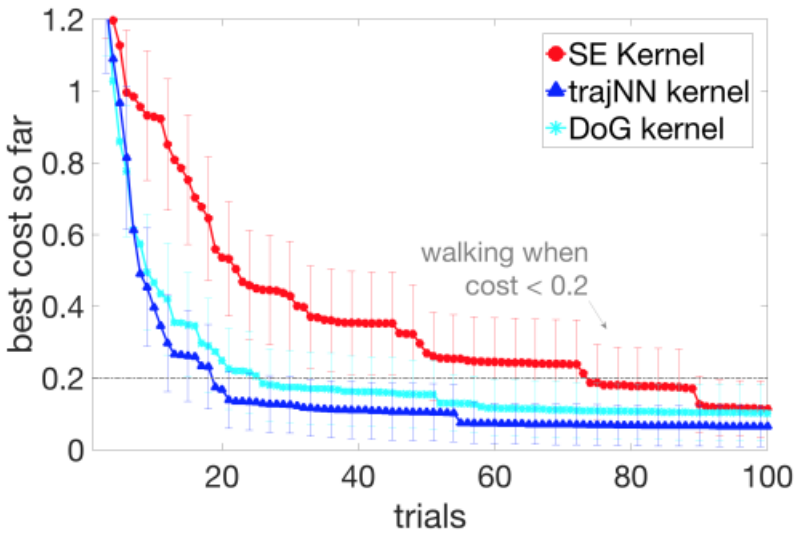


Figure 2-7: Comparison of some GP kernels that are shown to work well on a bipedal robot [29, Figure 7a].

which essentially measures the difference between the performance measure of each input $\boldsymbol{\theta}$ rather than difference between the inputs themselves. Simulated experiments on uneven terrain with $\sigma_k^2 = 1$ and $\boldsymbol{l} = 1$ show that using this type of kernel with BO significantly decreases the number of rollouts (from ∼100 to ∼30) compared to the typically used SE kernel. This is especially impressive for such a problem with 16 policy parameters, which is typically too many for BO to work well. In a subsequent study by the same authors [29], another means of designing the kernel, called trajNN, was developed using a NN, which is shown to perform comparably to DoG, with the added advantage of not being dependent on the cost. Like [28], [29] also uses short simulations to gather information to construct the kernel. [29] introduces a cost-agnostic (trajNN) kernel k_{trajNN}. This kernel is given by:

$$k_{trajNN}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \sigma_k^2 \exp\left\{-\frac{1}{2}(\phi_{NN}(\boldsymbol{\theta}_i) - \phi_{NN}(\boldsymbol{\theta}_j))^\top \text{diag}(\boldsymbol{l}_k)^{-2}(\phi_{NN}(\boldsymbol{\theta}_i) - \phi_{NN}(\boldsymbol{\theta}_j))\right\}, \tag{2-17}$$

where l_k is a scalar length scale hyperparameter and ϕ_{NN} is the NN output in the form of trajectory information from the simulation. The trajNN kernel is more versatile and simpler to design than k_{DoG} because it requires no domain knowledge. Figure 2-7 shows a comparison between the typical SE kernel, the hand-designed DoG kernel, and the learned trajNN kernel. From the figure it can be seen that learning with these more informative kernels does improve the speed of learning, but they also make the algorithm much more complex.

2-8-2 Acquisition functions

Between the three common acquisition functions, [27] finds that UCB yields the best results when tested on the bipedal robot Fox with spherical feet. In this study, the UCB method takes only about 30 trials to learn a fast, robust gait. PI requires twice as many trials to achieve comparable robustness to UCB, whereas EI requires at least 60 trials to reach a much lower robustness level. Another analysis of the three aforementioned functions shows that

UCB is clearly advantageous for making sure there is enough exploration of the parameter space [26]. The authors of [30] also use UCB on their hexapod robot, manually choosing β to tailor the exploration-exploitation tradeoff.

Rather than using nonlinear optimization to sample parameter sets to be evaluated by the acquisition function, Cully et. al. use a method called exhaustive search to pick the parameter set $\boldsymbol{\theta}$ for the next BO iteration [30]. Exhaustive search is useful if there is a relatively small, discrete number of parameter sets under consideration. When this is the case, instead of using an optimizer, exhaustive search simply uses the acquisition function to evaluate every possible $\boldsymbol{\theta}$ option on the GP, and chooses the best acquisition function performer as the next set to test. Since in this project the Pareto front will contain a relatively small number of individuals ($\sim$300), exhaustive search is a realistic and efficient way to ensure only the most promising Pareto points are chosen to evaluate during BO.

2-9 Conclusion

In summary, this thesis project employs a task priority-based control approach on the humanoid Talos robot. NSGA-II, which has been used successfully before for learning task parameters with two objective functions, will be used in this project to learn a Pareto front of task control parameters for many objectives at once. Then, for an approach that has the potential to take this Pareto front and apply it to new trajectories, BO is a viable choice because of its ability to learn in few rollouts: this gives the algorithm the potential to be applied on a real robot.

To this author's knowledge, the combination of learning a Pareto front for many trajectories at once, and then using it to bootstrap the search for a successful parameter set for new trajectories has not been previously validated. This approach combines modularity with data efficiency to be flexible and fast enough for real robot application.

Chapter 3

Multi-objective optimization

The first part of this project focuses on training a group of control parameter sets for many different goal trajectories at once, initially with no self-collision checking and then with self-collision checking. This is made possible by multi-objective optimization (MOO), which allows the learning of multiple separate objective functions at once. The resultant Pareto front of parameter sets contains a variety of different individuals, each of which performs differently on each trajectory. The Non-dominated Sorting Genetic Algorithm II (NSGA-II) algorithm yields a diverse Pareto set of control solutions that can then be analyzed to learn what control parameter values work well for different trajectories.

3-1 Methods

In this section, in order to highlight that this MOO approach can be applied generally to the problem of automatic tuning of control parameters, we first discuss the overarching theme of the method. Figure 3-1 shows the steps of the rollout and how the rollouts are used for NSGA-II to optimize the objectives $f_b(\boldsymbol{\theta})$.

3-1-1 Task formulation

In this work the parameter set that is learned by NSGA-II is composed of task priority-based control weights and gains as explained in Section 2-2. In task priority-based control, the choice of which tasks to consider greatly influences the performance of the robot. Here, the tasks used reflect the most important goals for accuracy and stability of the performed trajectories on the Talos. In our MOO formulation, there are two parameters per task that are optimized: the soft priority weight (SPW) which measures the importance of the task compared to the other tasks, and the convergence gain (CG), which defines the responsiveness of the controller to errors in task position and velocity.

The parameter set $\boldsymbol{\theta}$, consisting of these SPWs and CGs, captures the behavior of the controller. This parameter set is what will be optimized for each trajectory, and the Pareto front generated by MOO will contain many possible choices for $\boldsymbol{\theta}$.

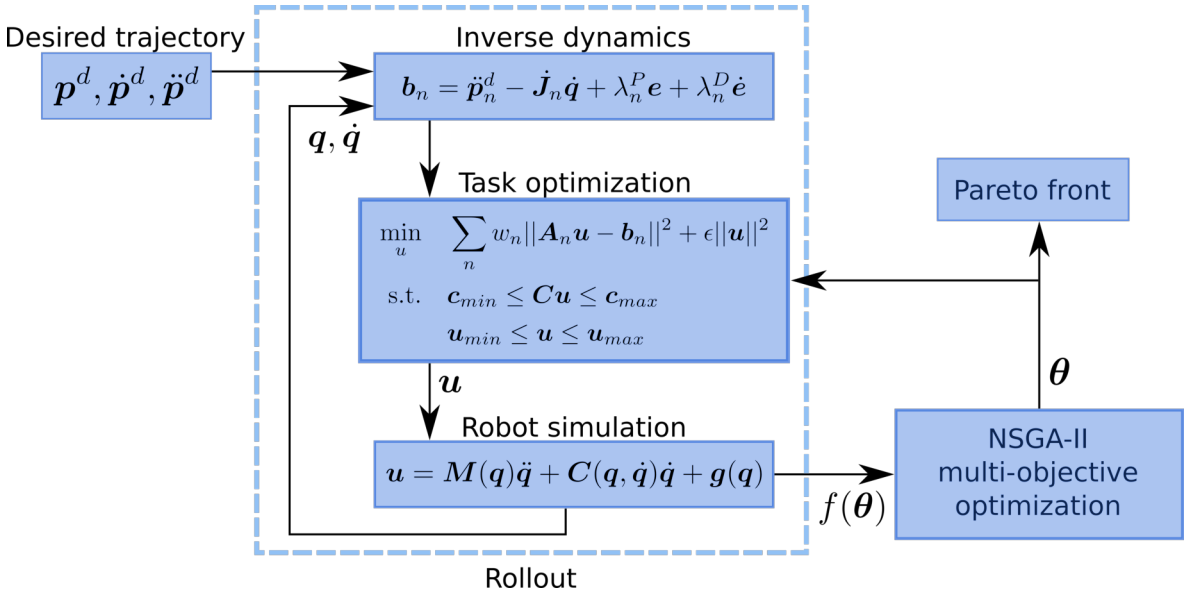


Figure 3-1: Diagram of the steps required to carry out NSGA-II optimization, starting with a goal trajectory and ending with a Pareto front of solutions.

3-1-2 Pareto front dimensions

Next, in order to generate the Pareto fronts, the quality of a given parameter set is measured by the objective functions. The main aspect of the NSGA-II implementation in this thesis that sets it apart from that of [6] is the objective function setup that is used. Penco et. al. use two objective functions $f_a(\boldsymbol{\theta})$ and $f_r(\boldsymbol{\theta})$, one which measures positional accuracy and the other which measures robot stability. Instead, in this thesis work, there is a separate objective function for each training trajectory chosen. This means that the resultant Pareto front is l-dimensional, where l is the number of objectives, corresponding in this case to the number of training trajectories used.

3-2 Experiments in simulation

In order to perform inverse dynamics task priority-based control, the Task Space Inverse Dynamics (TSID) library [32] was used on top of the Pinocchio framework [33] in C++.

The MOO algorithm, NSGA-II, was parallelized on a 256-core computer using the Sferes$_{v2}$ evolutionary algorithm framework [34]. Parallelization drastically reduces the amount of time required for this computationally intensive algorithm by allowing hundreds of simulated rollouts to be completed at once. Nonetheless, to complete a single NSGA-II run of 500 generations with eight objective functions and without self-collision checking (checking self-collisions increases the amount of computations and thus takes much longer) requires roughly 2 days to finish on the 256-core computer that was used. Each of the task priority weights and gains was optimized in the range of $[0, 2000]$, the same range used when hand-tuning the task parameters.

The first datasets were gathered without self-collision checking. This made it much easier for

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

Table 3-1: Symbol, description, SPW name and CG type for each optimized task.

the optimizer to find very low tracking error solutions, although most of these very 'accurate' solutions turned out to be infeasible due to the robot colliding with itself (such as the hand hitting the hip) during the movement. The next set of data was gathered with collision checking enabled, which resulted in lower accuracy (in some cases it is necessary to have collisions to follow the trajectory exactly), but the individuals in the resulting Pareto fronts were much more feasible for real-robot deployment.

3-2-1 Tasks used

As shown in Table 3-1, there are five tasks used to guide the movements of the Talos. This means that the parameter set optimized by NSGA-II has 10 elements, and is structured as:

$$\boldsymbol{\theta} = \begin{pmatrix} w_h & w_f & w_{CoM} & w_{to} & w_p & \lambda_h^P & \lambda_f^P & \lambda_{CoM}^P & \sigma_{to}^P & \mu_p^P \end{pmatrix}. \tag{3-1}$$

Table 3-1 also denotes the type of each task. The hand and foot tasks are Cartesian tasks which track both position and orientation of those robot parts. The center of mass (CoM) task, rather, is solely a position task in the x and y dimension of the CoM (the ground projection), while the torso task tracks solely the roll and pitch orientations of the robot's torso. Finally, the posture task tracks the cumulative joint angle difference of all the robot's joints from their start position.

3-2-2 Objective function

For our implementation, each objective function is a measure of the Cartesian accuracy of three selected tasks: $\mathcal{T}_h$, $\mathcal{T}_f$, and $\mathcal{T}_{CoM}$. The form of each of these objective functions is:

$$f_b = \begin{cases} \frac{1}{N_t} \sum_{t=0}^{T} \left(\left\| \boldsymbol{e}_t^{CoM} \right\| + \left\| \boldsymbol{e}_t^{h} \right\| + \left\| \boldsymbol{e}_t^{f} \right\| \right), & \text{if robot not fallen} \\ 1.0 \times 10^{10}, & \text{if robot fallen,} \end{cases} \tag{3-2}$$

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

Set 1	Set 2
Walk on spot	Squat
Clap	Touch ground
Lean and twist	Dance
Right arm reach	Lift

Table 3-2: Separation scheme of training trajectories for each of the two self-collision checking-enabled MOO sets.

3-2-3 Training trajectories

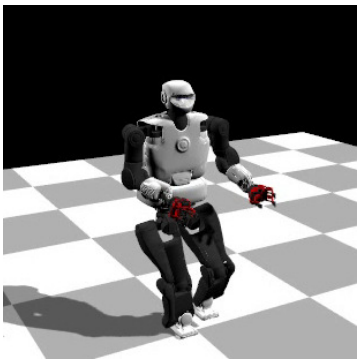
The more diverse the Pareto front, the more likely it is to transfer well to new trajectories. Since the trajectories used by NSGA-II to learn the Pareto front have a significant effect on this diversity, it is essential to choose training trajectories that use different parts of the robot and have different goals. For example, a walk-in-place training trajectory focuses on the feet and dynamic stability, a clapping trajectory focuses on hand tracking, and another trajectory which directs the robot to lean down and simulate picking up a box focuses on both hands and legs.

In all, there are eight training trajectories used, each of which has a duration of 20 s. These eight trajectories are listed in Figure 3-2 along with corresponding screenshots of the movements. This training set is evenly balanced with four handmade trajectories (walk on spot, squat, clap, and touch ground), for which the target positions were chosen by hand, and four trajectories retargeted from human motions recorded with the XSens MVN motion tracking suit [35]. The result of combining these trajectories is a suitably varied and diverse training set.

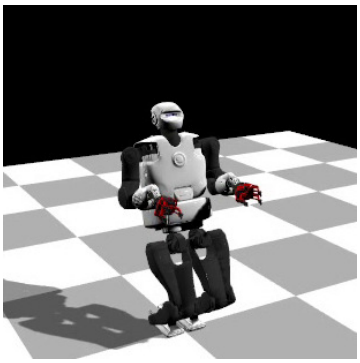
For tests which did not utilize self-collision checking, eight training trajectories and thus eight objectives were employed, whereas for the tests with self-collision checking, due to the increased computational load, only four training trajectories and objectives were used, and the number of generations was capped at 100. In order to glean optimized parameter information for all eight of the original training trajectories (to allow comparison with the parameter sets learned without collision checking), NSGA-II with self-collision checking was run twice with two different sets of training trajectories. Table 3-2 lists which trajectories were used as training sets in each of the two optimization setups. The choice of which trajectories to add to each set was done in order to make both sets reasonably diverse, and thus each set contains two handmade trajectories and two recorded trajectories.

3-2-4 Modified robot models

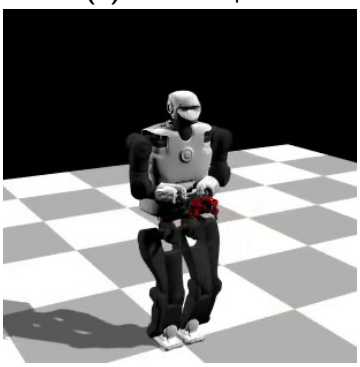
Since a simulation can never be an exact model of the real environment, one issue that is always present when training in simulation for deployment on a real robotic system is the *reality gap*, or the disparity between the simulation and real life. The obvious approach to solving this issue is to make the simulation as accurate (high-fidelity) to the real environment as possible. This can be achieved by two main approaches, either by improving the physical accuracy of the simulator, or by improving the accuracy of the robot and environment models. However, certain phenomena such as soft contacts or friction are too complex to estimate accurately, thus making a perfect model impossible.



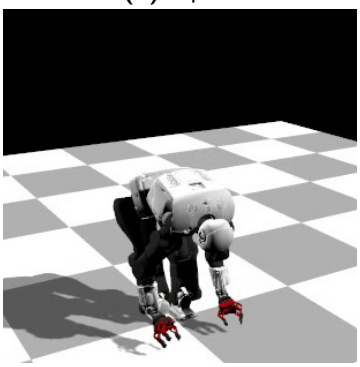
(a) Walk on spot.



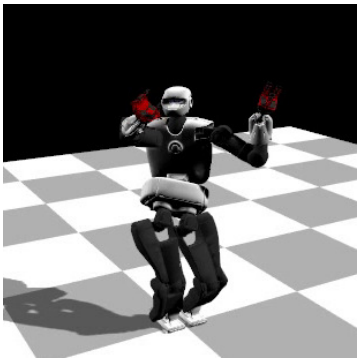
(b) Squat.



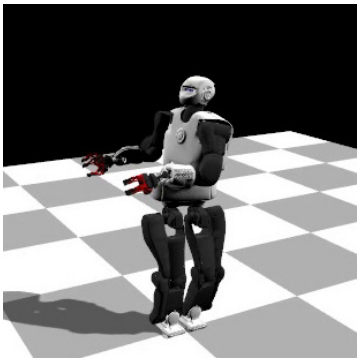
(c) Clap.



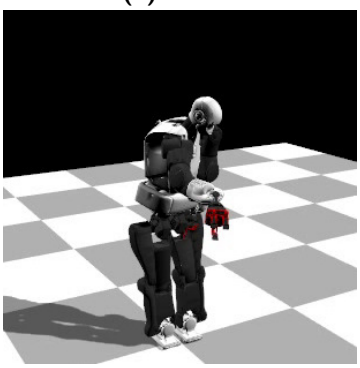
(d) Touch ground.



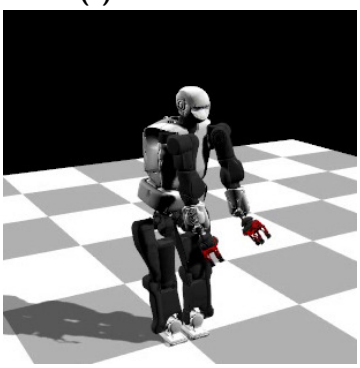
(e) Dance.



(f) Lean and twist.



(g) Right arm reach.



(h) Lift.

Figure 3-2: Training trajectories used during multi-objective optimization. Figs. 3-2a, 3-2b, 3-2c, and 3-2d are handmade motions, while the other four are recorded and retargeted from human motions.

Model name	Description
10% heavier	Robot mass uniformly scaled up by 10%
10% lighter	Robot mass uniformly scaled down by 10%
5 kg left shoulder	5 kg mass added to the left shoulder
5 kg right hand	5 kg mass added to the right hand end effector
5 kg backpack	5 kg mass added behind the center of the robot's torso

Table 3-3: Description of modified URDF models used for testing transferability.

In order to evaluate the transferability (i.e. the success in crossing the reality gap) of the Pareto solutions from NSGA-II, multiple modified models (URDF files) of the Talos robot were created. Since the real robot is likely to have a slightly different CoM than the model, and since its mass distribution greatly affects its stability and overall success, these modified models, summarized in Table 3-3, vary the mass to help determine whether this method yields transferable controller solutions.

3-2-5 Results and analysis without self-collision checking

Initially, NSGA-II was run without self-collision checking enabled. This decision was made in order to get a preliminary idea of the performance of this method, since the simulation is much less computationally intensive with no collision checking. For this part, all eight of the trajectories in Figure 3-2 were used as training trajectories. Therefore, there were eight total objective functions in the MOO.

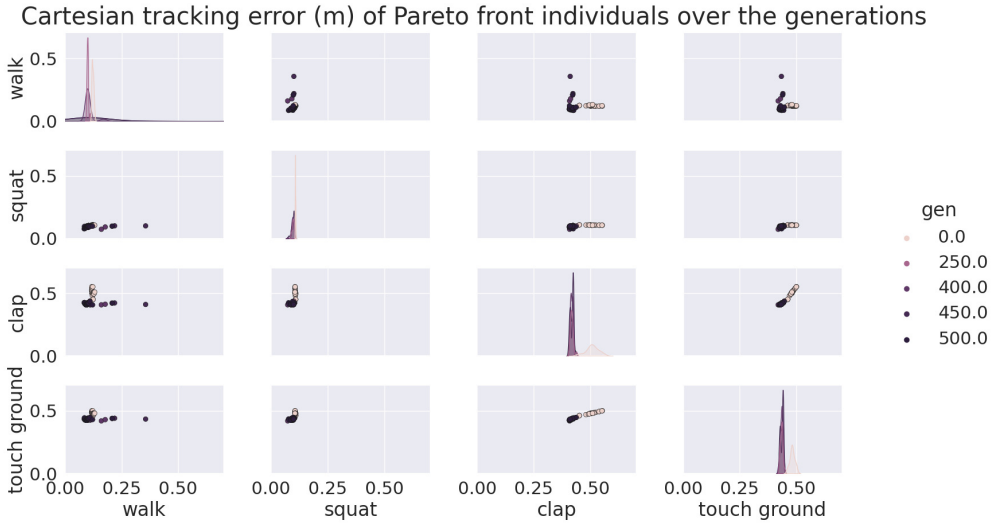
Number of generations

At the beginning, to avoid wasting time running the optimization for too long, the number of generations for which to run the optimization was chosen based on analysis of how the Pareto front changed over a given number of iterations. From this, it was determined that the Pareto front converges to a reasonably successful set of solutions by the 500^{th} generation, as shown in Figure 3-3. In the figure, the centers of the covariance plots change very little between the 450^{th} and 500^{th} generation, which means that by this point the individual solutions in the Pareto front are fairly constant. Extra training beyond this point would increase the risk of overfitting parameter solutions to the training trajectories, making the front less flexible for new trajectories. Consequently, NSGA-II was run for 500 generations for each dataset.

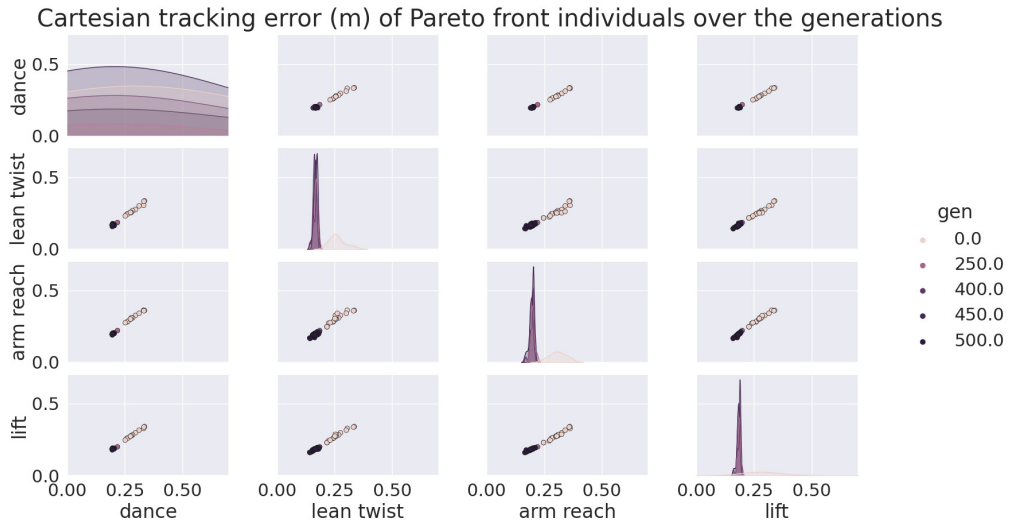
Learned parameter values

Once the number of generations was decided, five separate optimizations were run. The values that were learned for each parameter in $\boldsymbol{\theta}$ are very useful for understanding the control needs of different types of trajectories, so boxplots of these learned values are depicted in Figures 3-4 and 3-5.

From these figures it is clear that the best-performing parameter values vary widely depending on the goal trajectory in question. However, two constants for every trajectory are that the CoM weight w_{CoM} is uniformly close to the top of the allowed range, and the posture weight



(a) Plot of convergence of the Pareto front over 500 generations for handmade behaviors.



(b) Plot of convergence of the Pareto front over 500 generations for recorded behaviors.

Figure 3-3: These plots show that without self-collision checking the Pareto front converges to a fairly steady group of parameter sets by the time generation 500 is reached. Solutions which yield tracking error of more than 0.5 m can be considered unsuccessful and are not shown in order to keep the plots readable.

w_p is relatively close to zero for every trajectory. The values of w_{CoM} can be explained by the fact that the CoM position is the most important indicator of stability, which determines the success of a parameter set. Meanwhile, the posture task uses the robot's starting joint angles as a reference, which explains why the w_p values must be low: if the posture task receives a high priority, the robot cannot follow the trajectory since it will not move at all. The reason the posture task is included to begin with is that by discouraging large movements, it helps to make the robot's movements more efficient and less jerky.

It is also discernible from the figures that for some trajectories, there are certain parameters whose value does not matter much, such as the hand weight w_h for the dance trajectory (Figure 3-4). What this result shows is that for this particular trajectory, the robot's hand position is not important for stability, probably because during this motion the robot is in a very stable stance with its knees bent, so the hands are less likely to throw it off balance. However, it can also be seen from this boxplot that the median value is high, which shows that although the hand positions are not important for stability, they are important for accurate tracking of the trajectory. Conversely, for some trajectories, there are certain parameters that can only have a particular, exact value in order to work well. For instance, in Figure 3-4, the lift trajectory is the only one whose w_f boxplot has a tiny inner quartile range. This means all of the most successful parameter sets for this trajectory had a very specific w_f value of about 1500.

Some of the results are surprising, such as the fact that the hand weight is high for the squat trajectory. This can be explained by the fact that the hands balance the robot to avoid falling backward. Another unexpected result is that the touch ground trajectory requires a very high w_f, even though its feet do not need to move. This may be because the orientation, not the position, of the feet is crucial to keep the robot from falling. On the other hand, an expected result from Figure 3-5 is that all the CGs for robust trajectories are relatively low: these gains are measures of the controller's reactiveness, and the more reactive it is, the more risk the robot runs of becoming unstable and failing.

For these training results without self-collision checking, a video comparing the most accurate learned parameter set $\boldsymbol{\theta}$ for each trajectory to a generic or robust $\boldsymbol{\theta}$ is available at https://youtu.be/cnIo-aWCOcs. This video shows that there is a large disparity in the tracking performance of learned control parameters for a specific controller versus generic parameters. From the boxplot, we can also observe this compromise on accuracy that the robust controllers make: wherever the robust and trajectory-specific boxplots do not overlap, such as between the squat and robust λ^P_{CoM} boxplots, it means none of the robust controllers are in the top 50 (out of ∼1500 Pareto individuals total across the five datasets) highest performers. Clearly, a robust parameter set sacrifices accuracy in order to succeed on multiple trajectories.

Knowing the typical best-performing parameter values for different trajectories is useful for multiple applications, such as the design of better-performing hand-tuned parameter sets, or providing initial estimates in order to speed up the optimization.

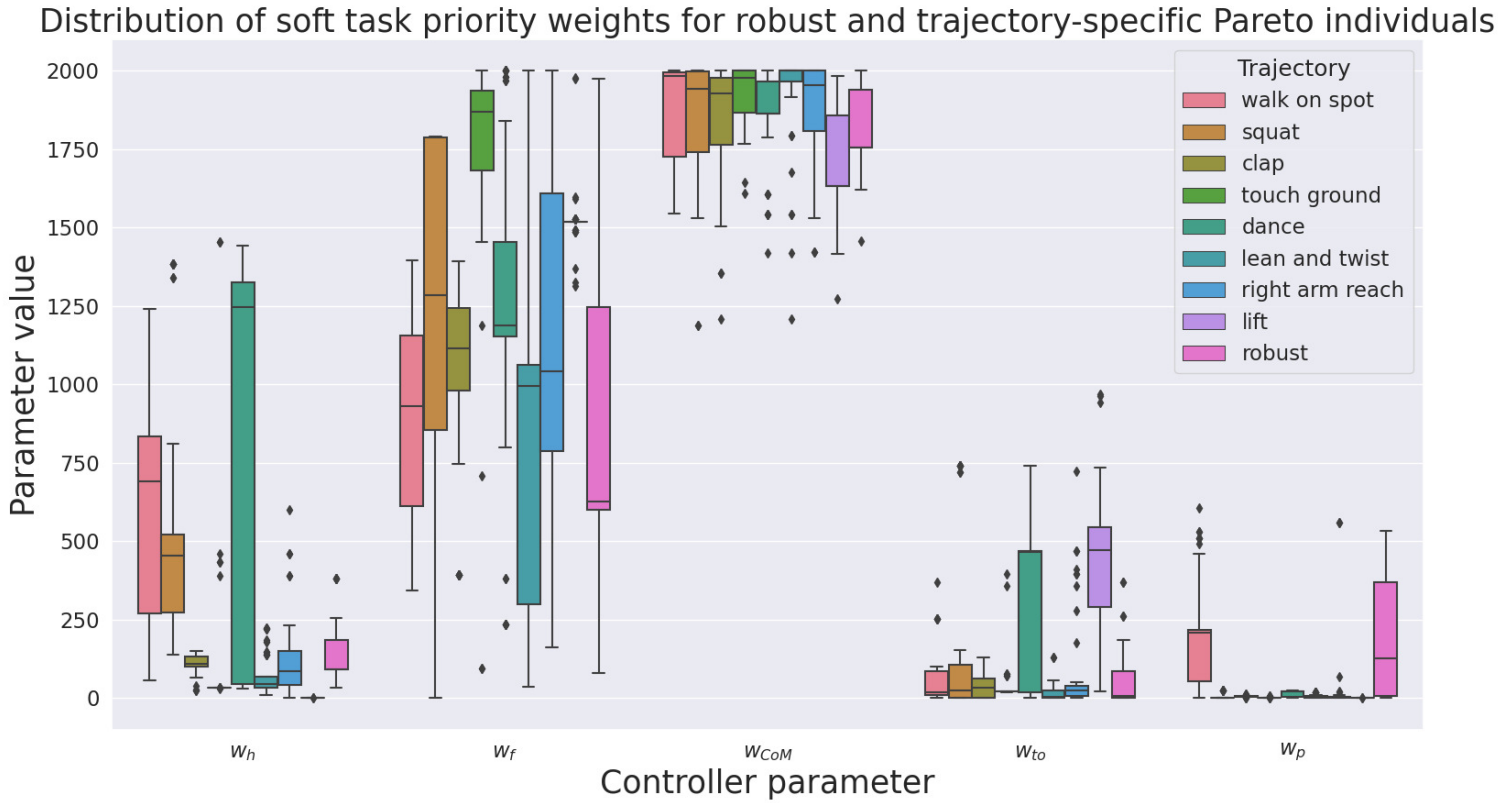


Figure 3-4: Comparison of top 50 best-performing learned SPWs for each training trajectory over five datasets without self-collision checking.

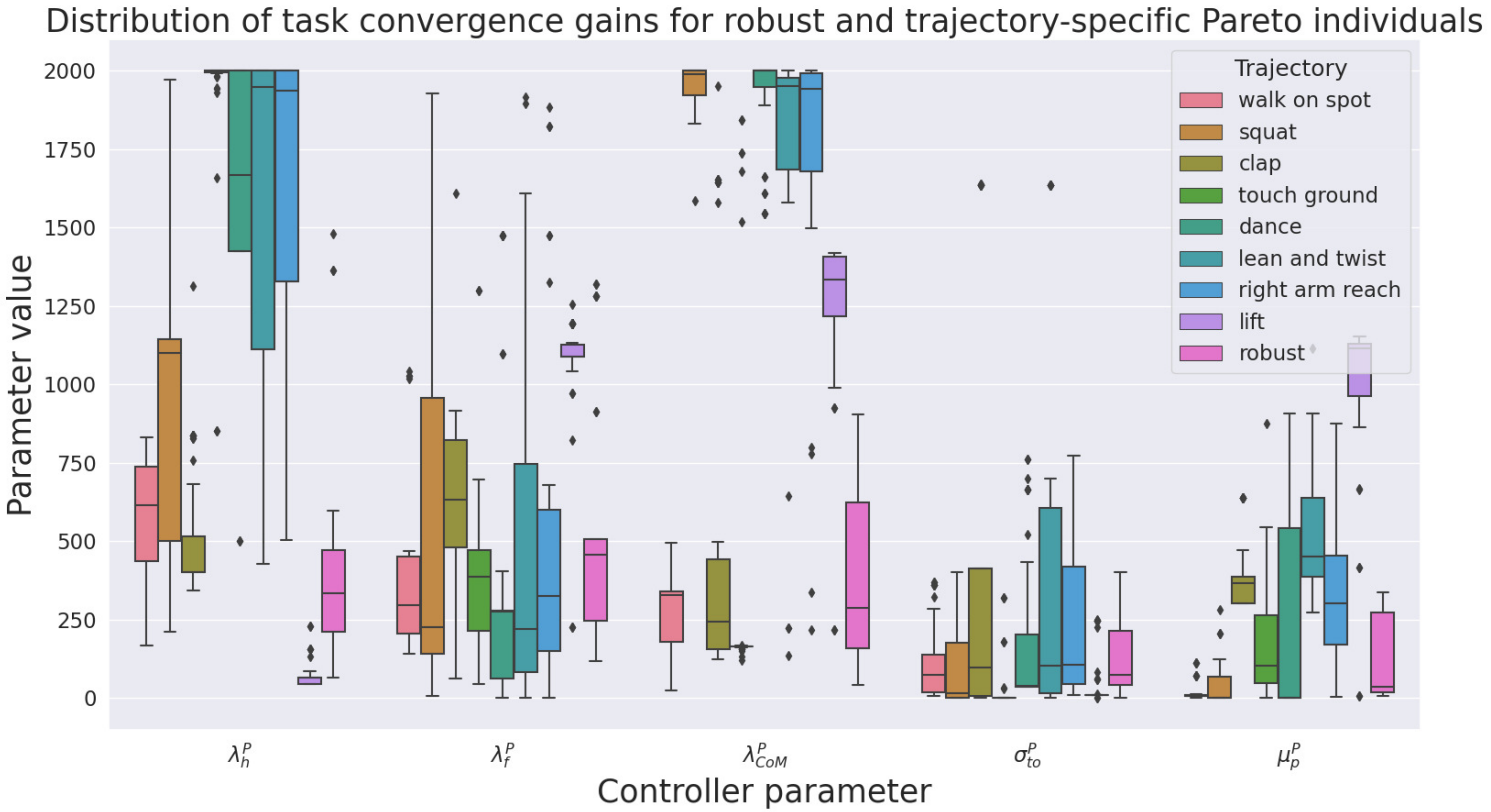


Figure 3-5: Comparison of top 50 best-performing learned CGs for each training trajectory over five datasets without self-collision checking.

Trajectory	Original	10 percent heavier	10 percent lighter	5 kg right hand	5 kg left shoulder	5 kg backpack
Walk on spot	**35.6%**	**16.2%**	**35.6%**	**16.8%**	**9.3%**	**4.8%**
Squat	**100%**	**99.7%**	**100%**	**99.7%**	**100%**	**100%**
Clap	33.5%	29.3%	31.1%	16.8%	21.9%	31.7%
Touch ground	53.6%	45.5%	56.6%	30.2%	40.7%	42.2%
Dance	45.2%	24.6%	31.1%	12.0%	29.0%	40.7%
Lean and twist	78.4%	74.9%	75.7%	35.0%	71.0%	86.2%
Right arm reach	**100%**	**100%**	**100%**	**99.1%**	**100%**	**100%**
Lift	88.6%	88.6%	89.2%	78.7%	88.9%	94.6%

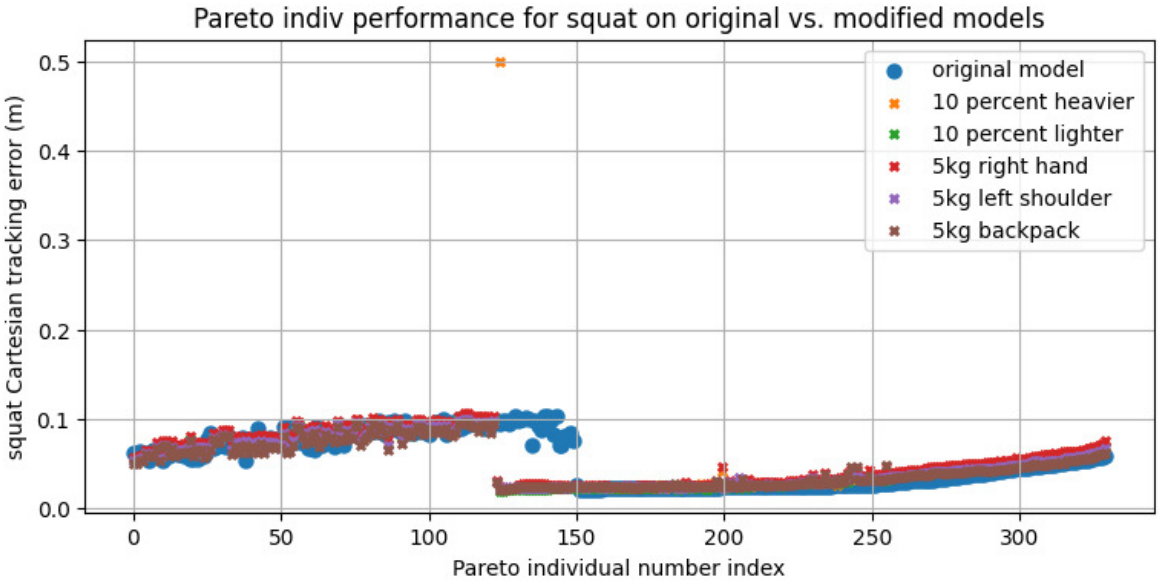
Table 3-4: Percent of total non-collision-checked Pareto individuals successful on each robot model.

Performance on modified models

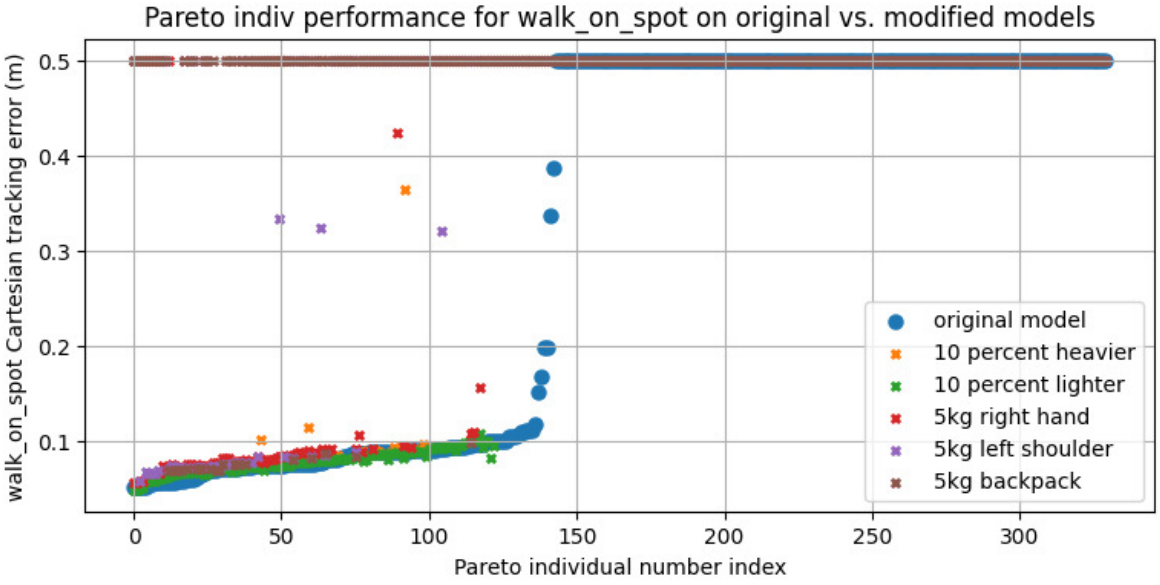
After analyzing the performance of the Pareto fronts on the original robot model, it was also evaluated on the modified models. Table 3-4 shows for an example dataset the proportion of the Pareto front individuals that succeed (do not fall) on the original and modified models. The information in this table indicates that for most of the trajectories, using a modified model does not have a large effect on whether the Pareto individuals fail. In fact, the squat and right arm reach trajectories are non-failing with almost every parameter set for every model. The walk on spot trajectory, on the other hand, does not succeed on very many Pareto individuals on the original model, and on the modified models its performance suffers even more. A reasonable explanation for this result may be that since this is the only trajectory that moves its feet off the ground, it is more susceptible to becoming unstable.

The modified model which affects the trajectories most negatively is the one which adds 5 kg to the right hand. This makes sense since it is the only modified model which adds extra weight at an end effector, which is further from the CoM and thus has a larger effect on the inertia of the robot. This result shows the usefulness of having a large number of parameter options in the form of a Pareto front, because a hand-tuned solution which works for a robot with no payload is unlikely to work for a robot holding an object. Analyzing the performance of the Pareto front on this type of modified model helps to weed out solutions that will not work when the robot works with payloads.

Despite the fact that the modified models affect the performance of the Pareto front, they do not affect its shape. This was verified by plotting the fitness versus Pareto individual number between the original model and the modified model. Figure 3-6 shows two examples of these plots, one for the squat trajectory, whose performance does not suffer on the modified models, and one for the walk on spot trajectory, which is severely affected by the model changes. From the plots, it is clear that although some parameter sets that succeed on the original model fail on the modified model, the overall shape of the Pareto front is similar. This means that the region of the Pareto front with the best Pareto individuals on the original model is the same on the modified model. Since the main difference between the simulation model and the real robot will likely be the distribution of mass, these results imply that there is no need to close the reality gap.



(a) Squat.



(b) Walk on spot.

Figure 3-6: Examples of the change in performance of a Pareto front between one robot model and another, without collision checking.

Trajectory	Hand-tuned (m)	Learned (m)
Walk on spot	**0.0923**	0.1020
Squat	0.1196	**0.0894**
Clap	0.4375	**0.4105**
Touch ground	0.4433	**0.4242**
Dance	0.1970	**0.1898**
Lean and twist	0.1889	**0.1597**
Right arm reach	0.1881	**0.1865**
Lift	0.1881	**0.1770**

Table 3-5: Average Cartesian task error achieved for each training trajectory by a hand-tuned controller versus the average objective score of the robust controllers learned without self-collision checking.

Evaluation of generic parameter sets

Finally, more analysis is done to show that this MOO method of generating control parameters is superior to hand-tuning not just in terms of labor-intensiveness, but also in terms of performance. Here, the average Cartesian task errors over all the robust parameter sets found by NSGA-II are compared to those of a parameter set (refer to Equation 3-1 for order of task weights and gains) hand-tuned by members of the LARSEN lab:

$$\boldsymbol{\theta} = \begin{pmatrix} 10.0 & 1000.0 & 1000.0 & 10.0 & 1.75 & 30.0 & 30.0 & 30.0 & 30.0 & 10.0 \end{pmatrix}.$$

Table 3-5 shows that for all but one trajectory, the learned parameter sets achieve better tracking error than the hand-tuned set. For the lone trajectory on which the learned sets do not perform better, the difference is very small: less than 1 cm. These results show that generating a Pareto front of control parameter sets not only produces accurate solutions for multiple specific trajectories, it also yields generic solutions that perform as well or better than hand-tuned ones, and require a fraction of the hands-on time to generate.

It can also be observed from Figures 3-4 and 3-5 that the actual mean parameter values learned for a robust controller are different from those in the hand-tuned set. Specifically, the value for w_f decreases drastically and the value for w_{CoM} increases drastically, while λ_h^P, λ_f^P, and λ_{CoM}^P also increase by a significant amount.

3-2-6 Results and analysis with self-collision checking

The optimized results from Section 3-2-5 are useful for finding solutions that accurately track the training trajectories. Unfortunately, these results are not safe to use on the physical Talos due to the risk of self-collision. Therefore, another experiment was conducted using NSGA-II, this time employing a self-collision checker in the simulation.

Learned parameter values

For the self-collision-checked datasets, due to the amount of time required to create them, there is only one of each, although undoubtedly running a larger number optimizations would

yield a fuller view of the parameter ranges that work well for each trajectory. These datasets, since they are only optimized with four training trajectories and for 100 generations each, only have about 100 Pareto individuals: the Set 1 Pareto front has 120 individuals and the Set 2 front has 110 individuals. For SPWs, Figures 3-7 and 3-8 show that just as for the parameters generated with no self-collision checking, w_{CoM} and w_p are, respectively, very high and very low for every trajectory. Interestingly, the hand weights w_h for the clap, lean and twist, right arm reach, and squat trajectories are much closer to the upper bound of 2000 in this collision-checked set, whereas without collision checking the hand weights were in the lower end of the allowable range. It makes sense that when trying to avoid self colliding, the robot places high importance on the position of the hands, as these are the parts that are responsible for most collisions. On the other hand, from non-collision-checked to collision-checked Pareto fronts, the w_h increases modestly for the walk, touch ground, dance, and lift trajectories, which do not move the hands much near the rest of the robot's body, and thus are at less risk of colliding.

The w_f and w_{to} values in the collision-checked sets are also in general slightly higher than those exhibited without collision checking. This may stem from avoidance of torso to thigh collisions. The typical value of the walk w_{to} is higher in Set 1 than in the non-collision-checked sets, but lower than that of the other trajectories in Set 1, and this is probably due to the fact that like the hands, the torso is not moving much in this trajectory and therefore is at low risk of colliding. Also, the squat and touch ground trajectories exhibit a large range for w_{to}, which means this task is not likely to cause self-collisions for these trajectories.

The CG boxplots in Figures 3-9 and 3-10 show that the learned parameters vary less between trajectories here than without collision checking. The λ^P_{CoM} values are universally very close to 2000, which is necessary to allow the robot to maintain stability. On the other hand, the torso and posture CGs are generally nearer to zero. It can be postulated that by making these robot parts less reactive, there are less jerky motions, lowering the risk of both self-collision and falling. The walk and squat trajectories buck these trends somewhat, and this is possibly due to the fact that a self-collision failure is much less likely than falling over for these trajectories (since the hands are not moving much). Except for those of the squat trajectory, the torso and posture CGs have a tiny range of values, meaning that these parameters need to be very precise not to cause collisions. The squat is the most stable and therefore 'easy' trajectory to find control parameters for, so for this reason it has a wider range of successful weights and gains. Conversely, λ^P_f values for each trajectory have a wide range and a relatively high median, suggesting that the specific value of this parameter does not have as large an influence on the possibility of failure, but that it does play a role in stabilizing the robot.

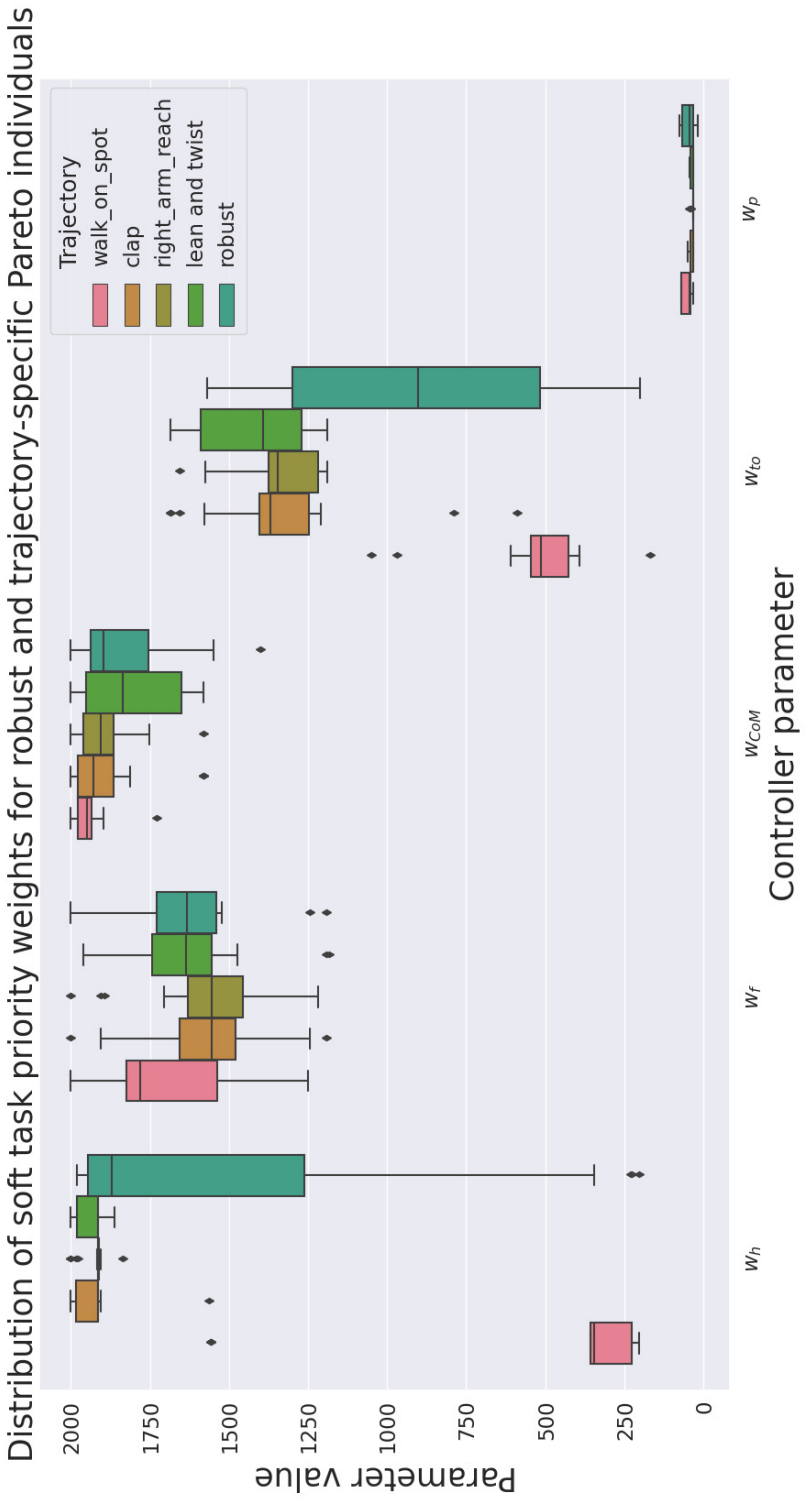


Figure 3-7: Comparison of top 20 best-performing learned SPWs for each training trajectory in Set 1, which enables self-collision checking.

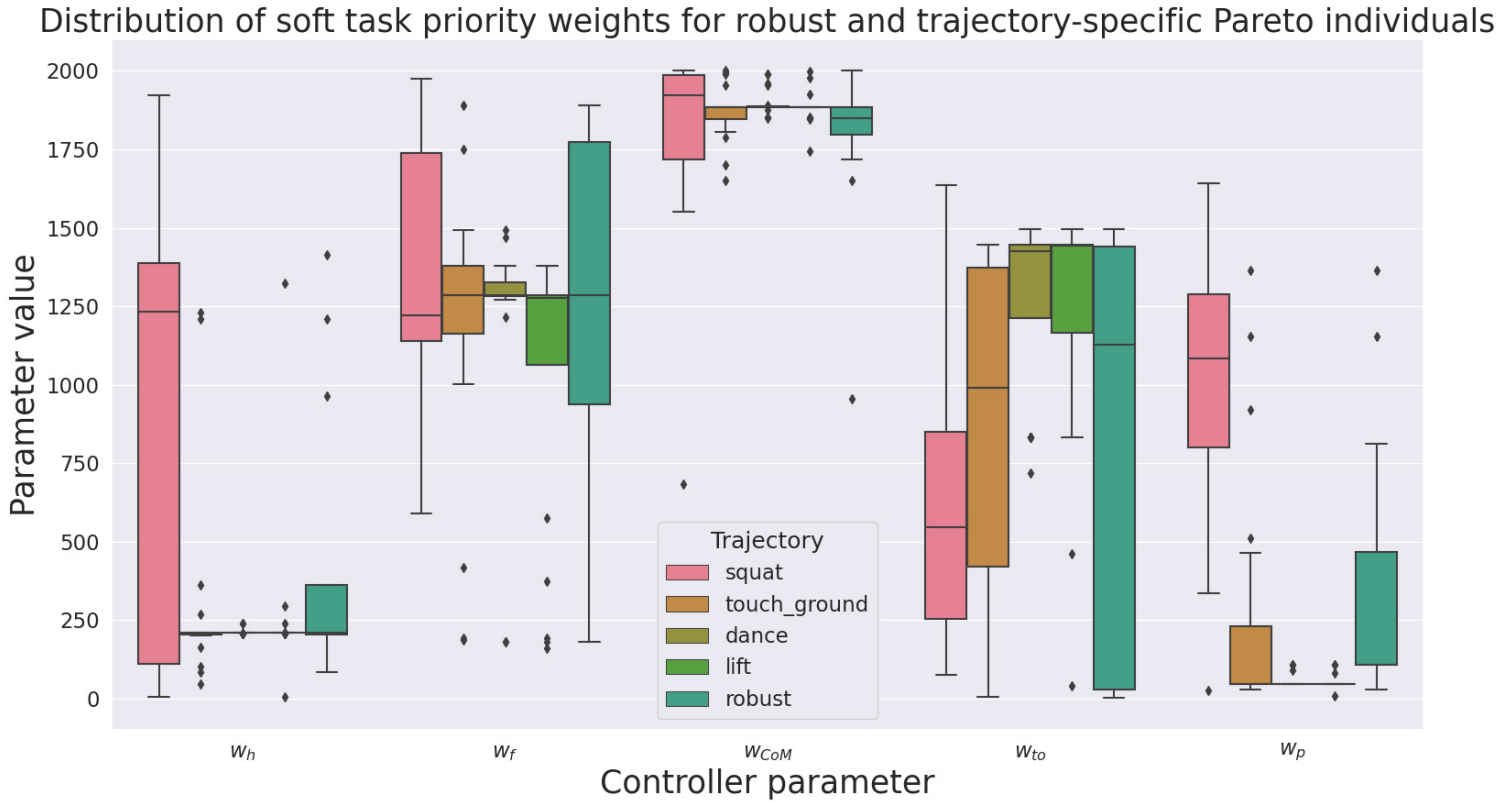


Figure 3-8: Comparison of top 20 best-performing learned SPWs for each training trajectory in Set 2, which enables self-collision checking.

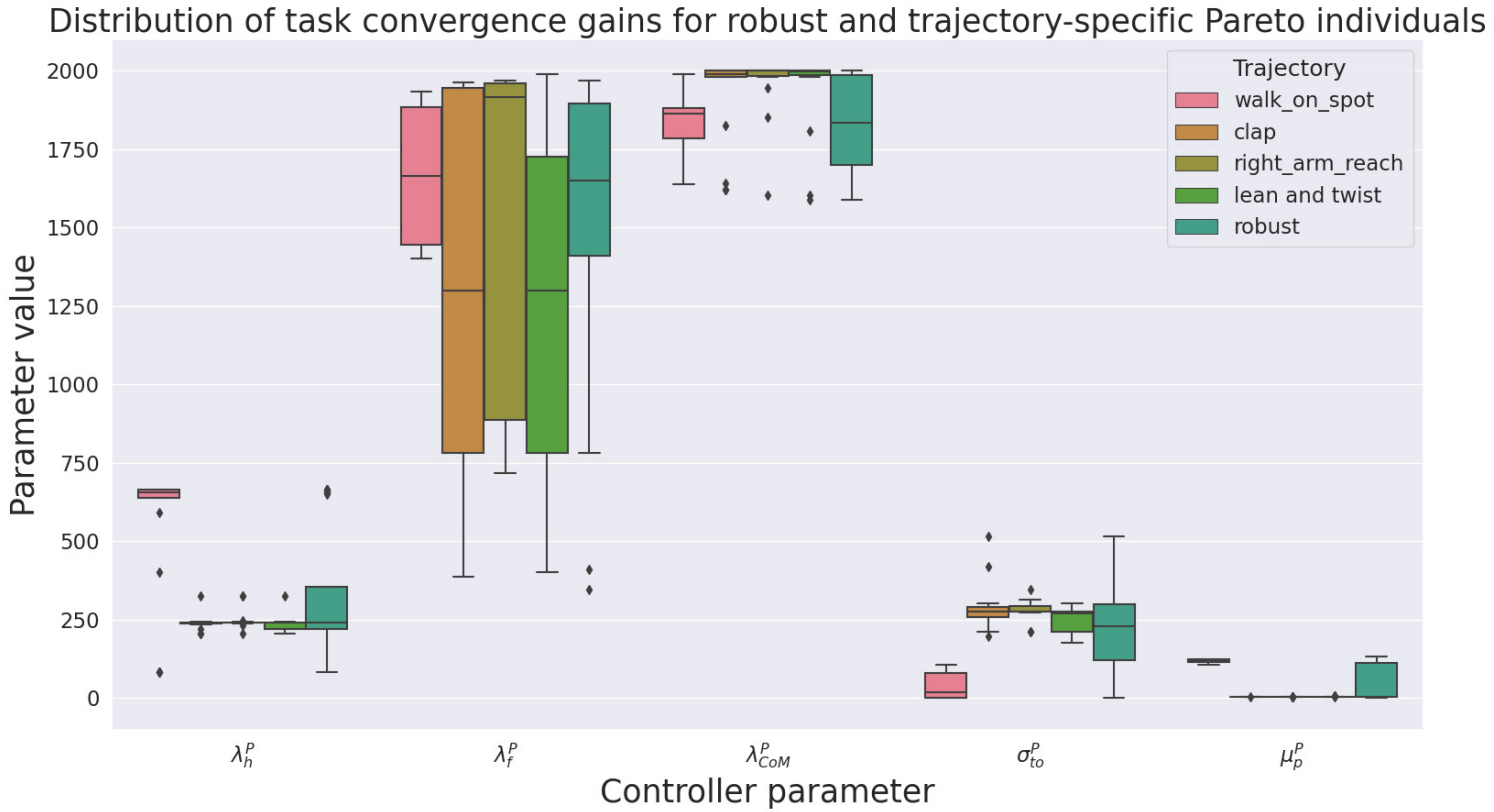


Figure 3-9: Comparison of top 20 best-performing learned CGs for each training trajectory in Set 1, which enables self-collision checking.

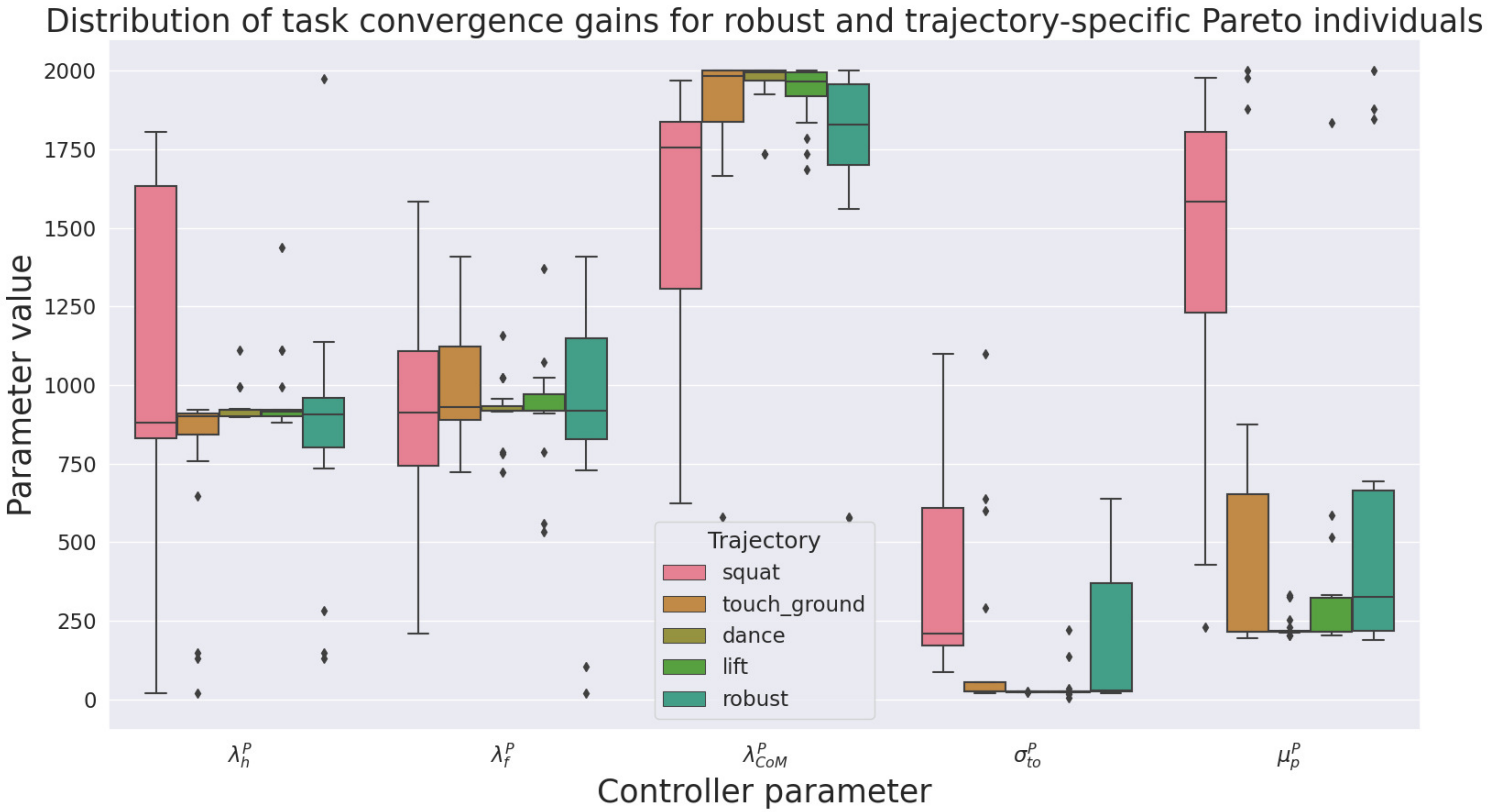


Figure 3-10: Comparison of top 20 best-performing learned CGs for each training trajectory in Set 2, which enables self-collision checking.

Trajectory	Original	10 percent heavier	10 percent lighter	5 kg right hand	5 kg left shoulder	5 kg backpack
Walk on spot	**94.2%**	**99.2%**	**98.3%**	**98.3%**	**82.5%**	**99.2%**
Clap	**47.5%**	**0.8%**	**1.7%**	**0.8%**	**0.8%**	**0.0%**
Lean and twist	65.0%	55.8%	71.7%	20.0%	33.3%	50.8%
Right arm reach	**86.7%**	**95.8%**	**100%**	**97.5%**	**97.5%**	**99.2%**
Squat	79.2	91.7	91.7	91.7	91.7	91.7
Touch ground	35.9	25.0	25.0	22.5	25.0	23.3
Dance	55.0	74.2	70.0	57.5	75.0	79.2
Lift	62.5	80.8	80.8	80.8	80.8	80.0

Table 3-6: Percent of total collision-checked Pareto individuals successful on each robot model.

Performance on modified models

After analyzing the learned parameter values, both Pareto fronts were tested on the same modified robot models as before. Table 3-6 shows that with self collision checking, unlike without, many of the trajectories exhibit less of a steep decrease in performance on new models. In fact, many of them (walk on spot, squat, dance, and lift) exhibit improved performance on most of the modified models. The walk on spot and right arm reach perform very well on every model, possibly because they are not moving the hands near the rest of the robot's body. On the other hand, the clap trajectory suffers much more markedly with self collision checking on the new models than it does without collision checking. This is probably due to the likelihood of the hands colliding with each other: on the original model the robot had enough trials to learn to avoid this, but changing the model throws off the delicate balance. It can also be observed from the Pareto plots of selected trajectories in Figure 3-11 that just as without collision checking, with collision checking the Pareto front tends to retain its general shape, even if a modified model causes more Pareto individuals to fail.

Evaluation of generic parameter sets

To evaluate whether NSGA-II learns solutions that are better than hand-tuning, once again the error of hand-tuned parameter sets is compared to that of learned sets. For self-collision-checked control, there are two hand-tuned parameter sets: one for handmade trajectories,

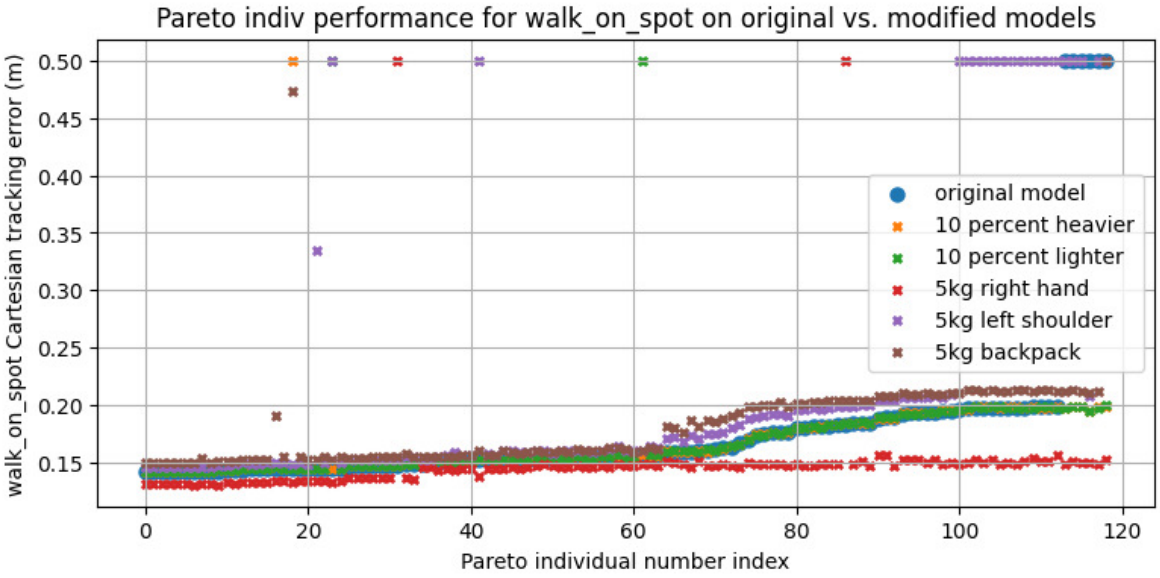
$$\boldsymbol{\theta} = \begin{pmatrix} 10 & 1000 & 1000 & 10 & 1.75 & 30 & 30 & 30 & 30 & 10 \end{pmatrix},$$

and one for recorded trajectories,

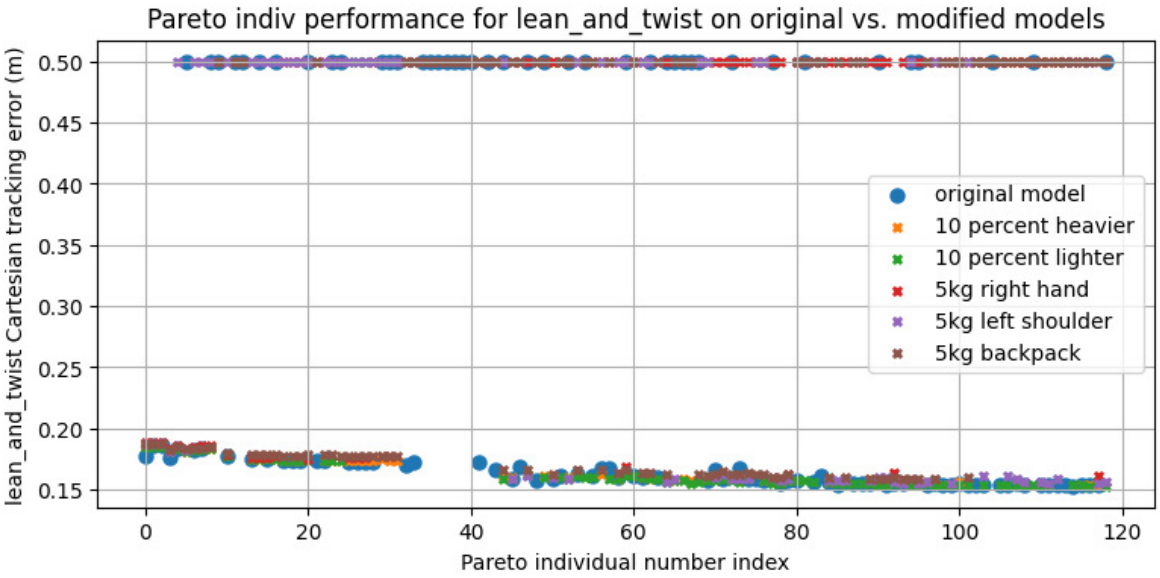
$$\boldsymbol{\theta} = \begin{pmatrix} 100 & 100 & 2000 & 1000 & 100.75 & 1000 & 30 & 1000 & 30 & 60 \end{pmatrix}.$$

The performances of the corresponding hand-tuned parameters are compared to the average error of the robust Pareto front solutions in Table 3-7. From this comparison we see that, as without collision checking, in most cases the learned robust parameter sets outperform the hand-tuned controller. This is especially clear for the touch ground trajectory, which was not included when the hand-tuning was performed. Because of this, the hand-tuned parameter set is unsuccessful on that trajectory.

Additionally, Figures 3-7, 3-8, 3-9, and 3-10 show that the actual mean parameter values learned for a robust controller are very different from those in the hand-tuned set, further illustrating the usefulness of optimization via this method.



(a) Walk on spot.



(b) Lean and twist.

Figure 3-11: Examples of the change in performance of a Pareto front between one robot model and another, with collision checking. Solutions with tracking error greater than 0.5 are labeled as failing solutions.

Trajectory	Hand-tuned (m)	Learned (m)
Walk on spot	0.1701	**0.1654**
Squat	0.2486	**0.2314**
Clap	0.5893	**0.1644**
Touch ground	1.0×10^{10} (self collides)	**0.3557**
Dance	**0.1936**	0.3234
Lean and twist	0.1845	**0.1649**
Right arm reach	0.1877	**0.1681**
Lift and twist	**0.2154**	0.2469

Table 3-7: Average Cartesian task error achieved for each training trajectory by a hand-tuned controller versus the average error of the robust controllers learned with self-collision checking.

Trajectory	No collision checking (m)	Collision checking enabled (m)
Walk on spot	**0.0426**	0.1417
Squat	**0.0195**	0.2303
Clap	0.3467	**0.1531**
Touch ground	0.2574	**0.1821**
Dance	**0.1447**	0.1822
Lean and twist	**0.1117**	0.1531
Right arm reach	**0.1376**	0.1532
Lift and twist	**0.1220**	0.1821

Table 3-8: Best Cartesian task error achieved for each training trajectory by Pareto fronts with and without self-collision checking.

3-2-7 Comparison

In the previous sections it is shown that while the learned parameters and consequent fitnesses are very different between non-collision-checked and collision-checked training configurations, NSGA-II is successful in finding a reasonably large set of well-performing options in both cases.

Table 3-8 shows that in most cases the tracking error is higher with self-collision checking than without. This is expected, since in effect, the robot's movements are more constrained in the collision checking case. However, self-collision checking is necessary for learning parameters that are safe to test on the real robot.

3-3 Experiments on the real robot

After evaluating the performance of the learned Pareto fronts in simulation, two of the trajectories were validated on the real Talos robot. This was done for two reasons: to see if it performed similarly to the simulation, and to compare a learned parameter set with the hand-tuned parameter sets. The two trajectories chosen for this test were the squat and dance, one which is a very stable, handmade trajectory, and the other which is a less stable motion recorded by a human. We executed each of the trajectories using both the hand-tuned parameters and the NSGA-II parameters learned with collision checking.

3-3-1 Parameter sets used

The hand-tuned parameters used for these tests correspond to those listed in Section 3-2-6, where the set for handmade trajectories is used for the squat and the set for recorded trajectories is used for the dance. In constrast, the learned parameters were chosen based on their success in the Gazebo simulator, which has higher fidelity than the simulator used for NSGA-II optimization. The control parameters with the best squat tracking error in the Set 2 Pareto front worked well in Gazebo. On the other hand, the Set 2 Pareto solutions that worked the best for the dance trajectory during training were too unstable and failed in Gazebo. Therefore, the learned parameter set we chose to use for the dance trajectory on the real robot was the seventh best in the Pareto front. The values of the chosen NSGA-II-generated parameter sets are listed below:

$$\boldsymbol{\theta}_{squat} = \begin{pmatrix} 1296.6 & 1241.3 & 1871.3 & 248.9 & 26.1 & 1591.1 & 916.9 & 1968.6 & 618.0 & 231.5 \end{pmatrix}$$
$$\boldsymbol{\theta}_{dance} = \begin{pmatrix} 207.9 & 192.1 & 1885.1 & 852.3 & 45.9 & 898.6 & 1074.7 & 1935.8 & 26.0 & 215.9 \end{pmatrix}.$$

3-3-2 Error measurement and calculation

A motion capture system was used to track the positions of the hands and feet of the robot in order to estimate their tracking error. The 3-D tracking data from this motion capture was compared to the reference positions sent to the robot over the course of each rollout. Comparing these two sets of information raises some issues, one of which is that the motion capture measurements need to be transformed to the same reference frame as the goal positions. To achieve this, Umeyama's method [36] was used, where matching points from each dataset are manually chosen and then an optimization problem is carried out to estimate the proper transformation. This optimization problem is given as:

$$\min_{c,\boldsymbol{R},\boldsymbol{t}} \quad \frac{1}{n}\sum_{m=1}^{n} ||y_m - (c\boldsymbol{R}x_m + \boldsymbol{t})||_2^2, \tag{3-3}$$

where c is the scaling factor, $\boldsymbol{R}$ is the rotation matrix, $\boldsymbol{t}$ is the translation vector, n is the number of point pairs used, m is the index of the point pair, and $\boldsymbol{x}_m$ and $\boldsymbol{y}_m$ are the input and output 3-D point vectors. In this way the result is a transformation from the reference frame of $\boldsymbol{x}_m$ to that of $\boldsymbol{y}_m$, such that the transformed $\boldsymbol{x}_m$ coordinates can be calculated using:

$$\boldsymbol{x}_{m,transform} = c\boldsymbol{R}\boldsymbol{x}_m + \boldsymbol{t}, \tag{3-4}$$

where $\boldsymbol{x}_{m,transform}$ is the transformed point.

Additionally, the motion capture measurements began before the robot started moving, while the reference positions begin when the robot starts the rollout. The starting point in the motion capture measurements had to be manually picked, which was somewhat difficult due to noise in the data. Noise makes it more difficult to choose accurate points for the transformation, and it also adds cumulatively to the error calculation. This problem was only significant for the squat motion, for which the feet stay in place and the hands barely move, so the noise is more noticeable. To mitigate this, a simple moving average (`smoothdata` function in MATLAB) was used to smooth it out.

Trajectory	Hand-tuned cost (m)		Learned cost (m)	
	Simulation	Reality	Simulation	Reality
Squat	0.1487	0.0638	0.1233	0.1391
Dance	0.1771	0.1179	0.1663	0.0896

Table 3-9: Comparison of modified cost f_{mod} between hand-tuned and learned parameter sets in simulation and reality.

The final issue with the measurements is that the motion capture and reference trajectory data were gathered using two different time step sizes. Since the motion capture operated at 120 Hz and the robot controller operates at 500 Hz, the error calculations were done at the largest common multiple of these frequencies: 20 Hz.

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

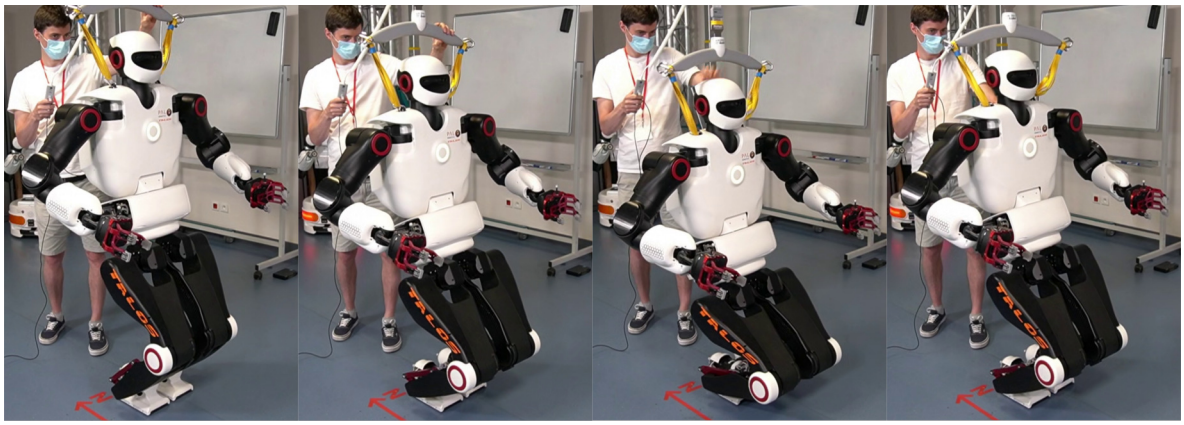
$$f_{mod} = \frac{1}{N_t} \sum_{t=0}^{T} \left(\left\| \boldsymbol{e}_t^{hands} \right\| + \left\| \boldsymbol{e}_t^{feet} \right\| \right), \tag{3-5}$$

After all of this manipulation of the data, Equation 3-5 was used to compute the cost of each experiment from the smoothed, transformed motion capture measurements.

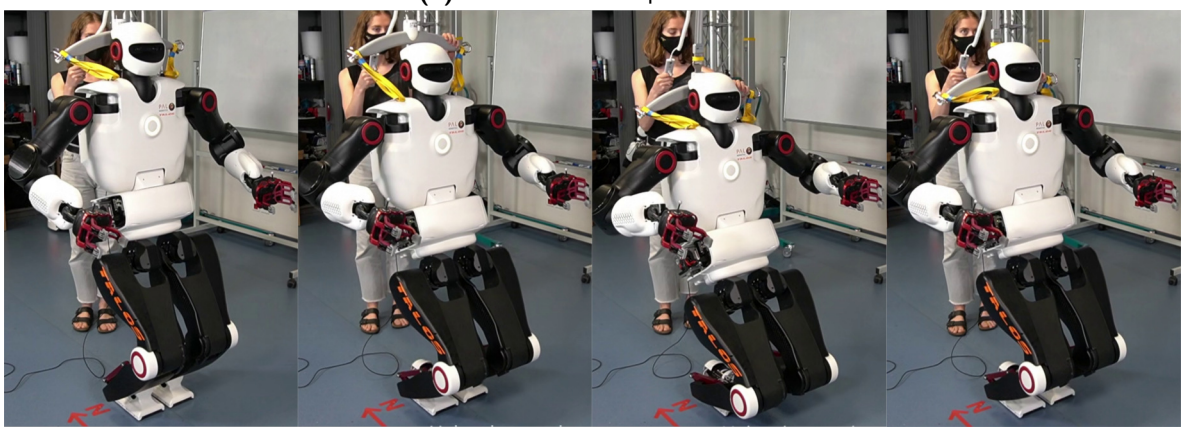
3-3-3 Results and analysis

The calculated costs of each of the experiments on the Talos are shown in Table 3-9, which compares the error computed via Equation 3-5 between the hand-tuned and learned parameter sets in both simulation and reality. Also, to visualize the tracking accuracy during the hand-tuned and learned parameter tests, Figures 3-13 and 3-14 compare the 3-D position over time of the right hand of the robot during each rollout to that of the goal trajectory. A video comparison of the learned and hand-tuned performance of these two trajectories on the Talos robot is available at: https://youtu.be/Cqe3ykzWGtY.

In simulation, we can see that the learned parameter sets show a modest improvement over the hand-tuned sets for both trajectories in terms of modified cost f_{mod}. In contrast, on the real robot, the results are less clear. From watching the rollout of the squat trajectory, it can be observed that with the hand-tuned controller the robot exhibits shaking in the hands as well as some movement of the wrists. Instead, the learned squat parameters seemed to work better: with the learned parameters the robot's wrists only moved in the range of about 8 mm back and forth during the motion, which is much less than they did with the hand-tuned parameters (see Figure 3-13), and this made the whole movement smoother and more stable. Figure 3-12 shows that the hands move much less in the world frame (compare their height against the background) with the learned parameters than with the hand-tuned parameters. Because of this, it is surprising that the error calculated for the hand-tuned parameters is less than half that of the learned parameter set. It can be observed from the trajectory of the right hand in Figure 3-13 that the learned squat data has much more visible noise than the other



(a) With hand-tuned parameters.



(b) With learned parameters.

Figure 3-12: Comparison of squat trajectory performance between hand-tuned and learned controllers on the Talos robot.

measurements. This is because the entire trajectory is within such a small range that the amplitude of the noise comprises a large percent of that range. The large proportion of noise made it difficult to choose ideal points for transforming the frame, which could contribute to a larger calculated cost.

For the dance trajectory, the calculated cost is about 3 cm less for the learned parameter set than for the hand-tuned set. It is shown in Figure 3-14 that the motions track the shape of the reference trajectory fairly closely and smoothly. The costs for the dance are likely to be more trustworthy than those for the squat, since the larger range of motion (compare bounds of squat and dance plots in Figures 3-13 and 3-14) implies two things: that it was easier to choose accurate points for the transform, and that the proportion of the error caused by sensor noise was much lower than it was for the squat. These two facts mean that the calculated costs for this trajectory have a lower percent of error from transform inaccuracies and noise.

Interestingly, the costs calculated from the real robot experiments are in general smaller than those found in simulation. One possible explanation for this, other than error introduced by data manipulation, is that the real robot has slightly different properties than the model used

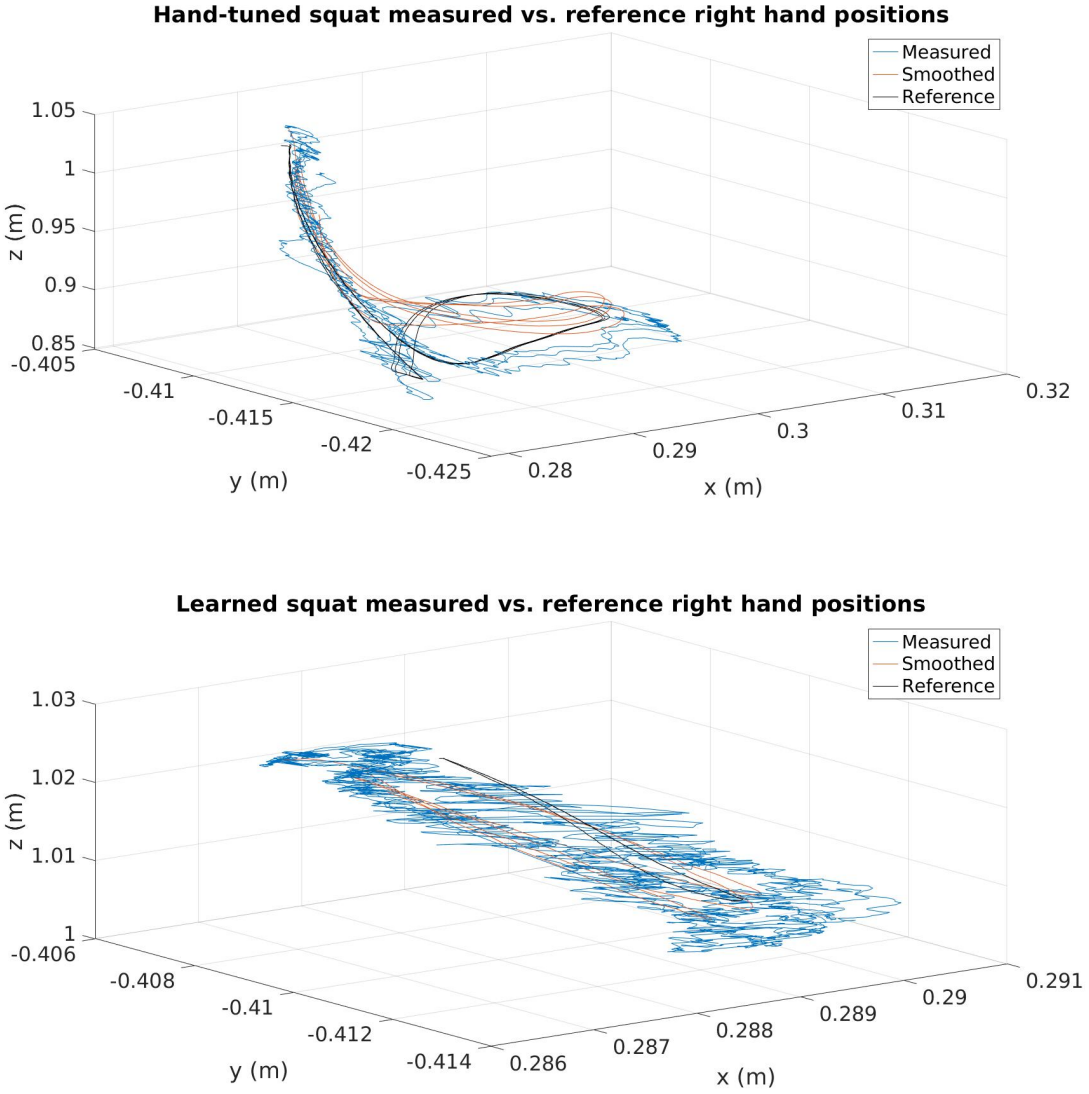


Figure 3-13: 3-D plots of the raw and smoothed measured squat trajectories vs. the reference trajectory for the right hand of the Talos robot.

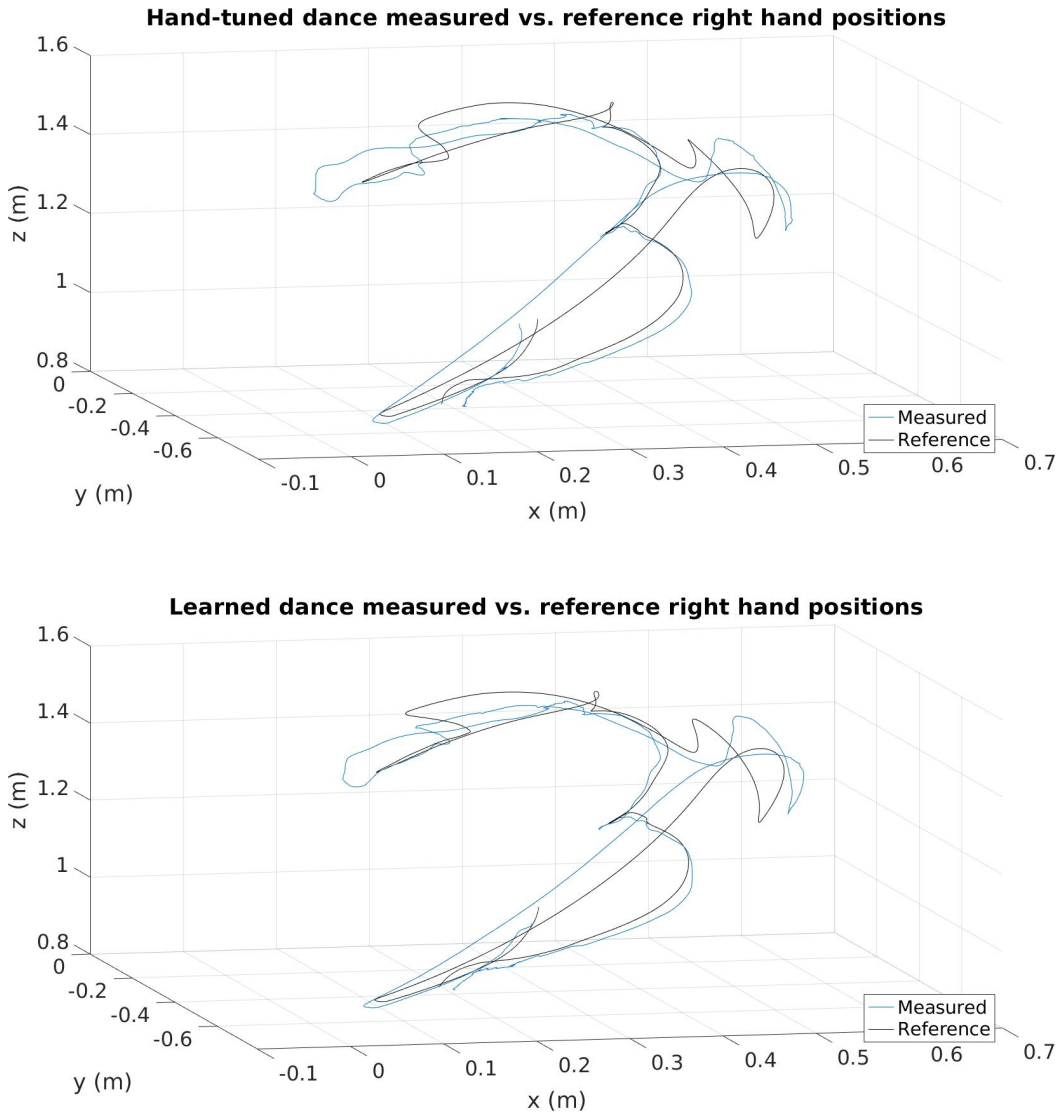


Figure 3-14: 3-D plots of the measured dance trajectories vs. the reference trajectory for the right hand of the Talos robot.

in simulation.

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

3-4 Conclusion

There are a number of takeaways from the results of the experiments explained in this chapter. Analyzing the Pareto fronts produced by NSGA-II allows the comparison between robust and specific parameter sets, between self-collision checking and no self-collision checking, between hand-tuned and learned parameters, and between different robot models. For instance, it is useful to know which range a given parameter must be in in order to yield a successful controller because this information can be used as an initial guess to speed up further optimization. It is also important to recognize the sacrifice in accuracy that must be made by the robot to avoid self-colliding. However, perhaps the most surprising result found in these experiments is that the Pareto fronts do not change shape when applied to different robot models. This is a positive result because it means no extra steps need to be taken between simulation and reality: the best individuals in a simulation-trained Pareto front are likely to also be the best on the real robot. In fact, the simulation-trained Pareto solutions tested on the real robot are shown to perform well.

Chapter 4

Bayesian optimization

The second part of this thesis project is based on using the Bayesian optimization (BO) algorithm both by itself as a comparison to the results from Non-dominated Sorting Genetic Algorithm II (NSGA-II), and as a complement to NSGA-II to bootstrap the learning. As a complement, BO is used to transfer the Pareto fronts to new trajectories. Since the goal is for this procedure to be achievable on the real Talos, for the Pareto front-based BO, only the robot-safe Pareto sets trained with self-collision checking are used.

4-1 Methods

In this work, BO was conducted in two ways: directly in continuous parameter space and along the previously generated Pareto fronts from NSGA-II results. Direct BO was used mainly as a baseline comparison. For the second and main method, Figure 4-1 shows: how the Pareto front and new trajectory are used as input, what the rollout process is, and that the final result of BO is a single optimal parameter set that works well for the new trajectory.

This implementation of Pareto-based BO used the optimized Pareto front as a search space, therefore the parameter space was finite and easily searchable. This initialization strategy is similar to that used in [30], where the initialization contained prior information in the form of a behavior-performance map that was pre-computed in simulation. The work in this thesis differs from their approach in the way that the map (Pareto front in this case) is generated. Here, since the parameter space is easily searchable, instead of using an optimizer (such as a nonlinear optimizer or Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14]), an exhaustive search algorithm similar to that used in [30] was employed. The mean function $\mu(\boldsymbol{\theta})$ of the Gaussian process (GP), which has a significant effect on the performance of the optimization, is initialized using the average of the fitnesses from the NSGA-II train trajectories for each Pareto individual. The Matérn $\frac{5}{2}$ kernel function in Equation 2-13 is used to represent the GP's covariance $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$.

The Upper Confidence Bound (UCB) acquisition was chosen because of the clear results in the literature indicating its superiority over other common functions. Additionally, for

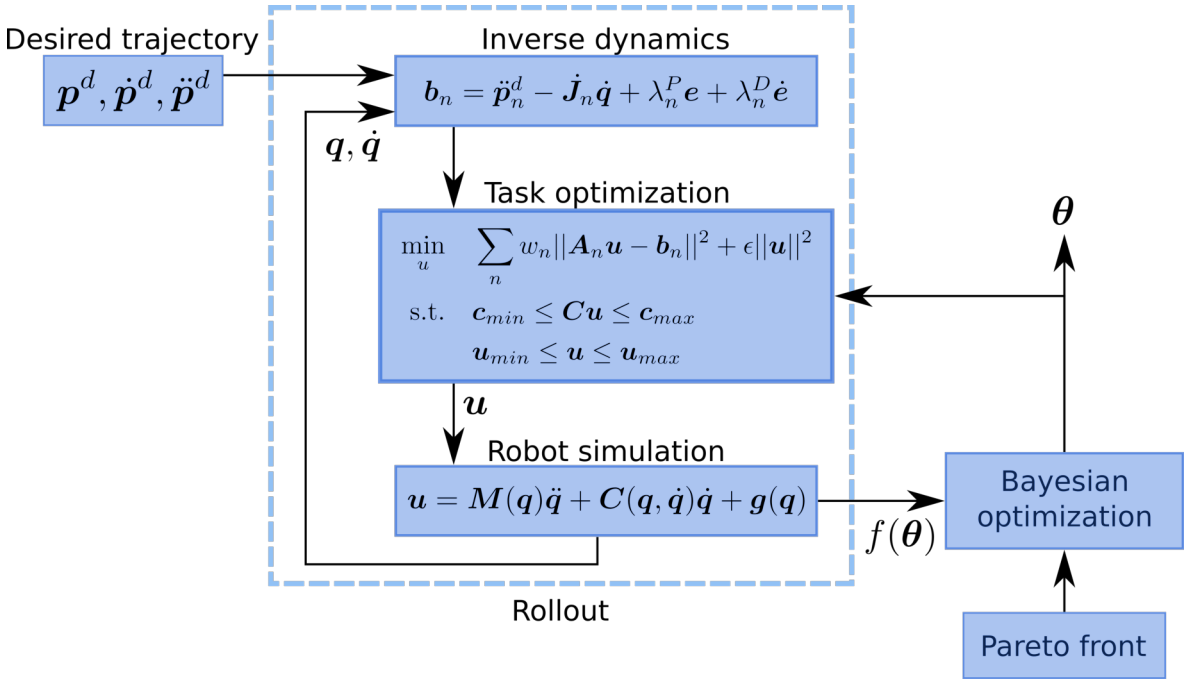


Figure 4-1: Diagram of the steps taken to optimize over a Pareto front with BO.

consistency the same objective function (Equation 3-2) adopted for NSGA-II was also used for BO to evaluate the success of new trajectories.

4-2 Experiments in simulation

BO was tested in two ways: within the continuous parameter search space as a comparison to other methods, and along the Pareto fronts generated from multi-objective optimization (MOO) to allow the robot to learn to follow new trajectories, on which the Pareto front was not trained. In the Pareto-based BO experiments, since the goal was to learn a new trajectory using a Pareto front which was not trained for that specific trajectory, the Set 1 trajectories were learned by searching along the Set 2 Pareto front, while the Set 2 trajectories were learned by searching along the Set 1 Pareto front (refer to Table 3-2).

4-2-1 Algorithm settings

For both of the ways in which BO is used, the settings of the algorithm affect its performance. The BO algorithm contains several variable parts and hyperparameters, each of which has a different effect.

One of these parts is the initial mean function for the GP. The choice of initial mean $\mu_0(\boldsymbol{\theta})$ determines whether the optimizer is optimistic or pessimistic, which influences its ability to find good solutions. In order to determine which of these approaches is better, direct BO is evaluated both with an initial mean of zero, which is very optimistic, and with an initial mean of -1.0, which is somewhat pessimistic since all successful objective scores will be greater

Combination name	$\mu_0(\boldsymbol{\theta})$	l_b	β
C1	-1.0/ avg w/ failing sols	1.0	0.5
C2	0.0/ avg w/out failing sols	1.0	0.5
C3	0.0/ avg w/out failing sols	5.0	0.5
C4	0.0/ avg w/out failing sols	1.0	2.0

Table 4-1: List of setting combinations tested for direct and Pareto-based BO.

than this value. On the other hand, since Pareto front-based BO has access to the Pareto solutions, instead of a constant mean, its optimistic initial mean is an average of the NSGA-II-trained fitness values over all training trajectories for each individual, ignoring failing fitness values. In contrast, the pessimistic initial mean for Pareto-based BO is the same average of the NSGA-II-trained fitness values, including failing solutions in the calculation which bring down the average significantly. When the initial mean estimate is pessimistic, the algorithm favors exploitation over exploration: since most of the parameter sets it tests will perform well compared to the initial mean, it will have little impetus to test other areas of the parameter space. On the other hand, when the initial mean is optimistic, the algorithm tends to explore more, since the untested areas of the parameter space retain a high UCB value.

Another tunable hyperparameter is the smoothness hyperparameter $l_b > 0$ of the GP kernel (Equation 2-13). Changing this hyperparameter affects how much the estimated expected returns $\hat{J}(\boldsymbol{\theta})$ of parameter sets near the sampled one are changed. The larger l_b is, the wider the spread of updates to the GP.

The last part of the BO algorithm that will be examined is the exploration hyperparameter $\beta > 0$ of the UCB acquisition function. This parameter governs how much the sampling will exploit promising regions of the parameter space versus exploring more of the unsearched space. A larger value of β results in more exploration, while a smaller value causes more exploitation.

In order to evaluate which of these hyperparameters and mean initialization schemes work best, the combinations are given the names C1-4 and the details for each is listed in Table 4-1. These combinations are tested for both direct and Pareto-based BO. In the upcoming plots, failing costs are scaled from 1.0×10^{10} to 0.7 for readability. Furthermore, each algorithm is run for 120 iterations. This is because the Set 1 Pareto front has 120 individuals, and the Set 2 front has only 110. Since random search along the Pareto front will certainly find the best solution by the time it has tried every Pareto individual, and since the BO approach is useless if it is less successful than random search, there is no need to see how BO performs beyond this number of iterations.

4-2-2 Results and analysis of direct BO

The first experiment conducted with BO in this thesis is with direct BO, which searches the entire allowable parameter space for solutions. The reason these tests are done is to have a competing algorithm to compare both to the results of NSGA-II and those of the Pareto-based BO approach, as well as to gain a better understanding of the effects of varying the aforementioned hyperparameters.

Comparison to NSGA-II

Table 4-2 compares the best tracking errors for each trajectory found within the NSGA-II Pareto fronts with the best tracking errors achieved by the direct BO setup (both with self-collision checking enabled). The results show that for all except the squat trajectory, NSGA-II far outperforms direct BO. This shows that although NSGA-II requires much more computation time, it is the superior method for generating highly successful parameter sets. The table also summarizes which BO hyperparameter combinations performed the best and how many iterations of BO it took to find the best cost. Both in the table and in the plots of Figure 4-2, it is apparent that the trajectory which the algorithm learns changes which settings work best.

Effects of changing smoothness hyperparameter

We can also see that different hyperparameter combinations learn faster than others. For example, in most of the plots of Figure 4-2, C3 finds a good (not necessarily optimal) solution earlier than C2. A larger l_b value means that when a parameter set is tested, the GP updates a larger range of the sets near it. So, a larger value of this smoothness hyperparameter should work well when the parameter sets in each region of parameter space perform similarly to each other. This is generally the case, except in regions where the parameters cause instability; in these areas of the parameter space, a tiny change made to a single parameter could cause the robot to fail. This appears to be the situation for the walk, touch ground, and lift trajectories, where C3 learns more slowly than the others. Maybe this is because the robot's center of mass (CoM) moves much more when shifting its weight from side to side or bending over, and thus its stability is more delicate.

Optimistic vs. pessimistic initial mean function

Between C1 and C2 the former learns faster in some trajectories (this is easy to see in Figures 4-2a and 4-2c for example), but the latter learns better parameters overall. The pessimistic mean initialization of C1, by assuming that all parameter sets will perform badly, encourages exploitation over exploration. This is because when it tests a point in parameter space, that set will likely have a cost of less than 1, therefore that becomes the only region of the search space which looks promising compared to the rest. Most of the time, especially when searching in the continuous parameter space, parameters nearby will indeed perform similarly well. However, a pessimistic mean initialization is more likely to get stuck in a local optimum, unlike an optimistic initialization, which has the opposite effect by prioritizing exploration. The trajectories for which the pessimistic C1 actually performs better than the optimistic C2 (squat and dance) are likely to have all the best parameter solutions clustered in a certain area of the parameter space.

Effects of changing exploration hyperparameter

Between C2 and C4, increasing the value of the exploration hyperparameter α pushes the algorithm to sample more often from unexplored areas of the search space. This is because

Trajectory	NSGA-II (m)	Direct BO (m)	Best direct BO hyperparameter combo
Walk on spot	**0.1417**	0.1561	C4
Squat	0.2303	**0.2141**	C1
Clap	**0.1531**	0.5039	C2
Touch ground	**0.1821**	0.4678	C2
Dance	**0.1822**	0.2357	C1
Lean and twist	**0.1531**	0.2035	C2
Right arm reach	**0.1532**	0.2390	C2
Lift	**0.1821**	0.2088	C3

Table 4-2: Best Cartesian task error achieved for each training trajectory by Pareto-based versus direct BO with self-collision checking.

a larger α increases the covariance of the GP, meaning that unexplored areas, which already have larger covariance than highly trafficked areas, are more likely to score well with UCB. For almost all of the trajectories, C4 learns faster than C2, but cannot find comparably successful parameter sets. An α value of 2 may increase the covariance so much that the algorithm can barely exploit at all, which prevents it from improving over time.

This study of the effects of changes to certain hyperparameters for direct BO is useful because it informs us about the characteristics of the successful parameter areas for each of these trajectories.

4-2-3 Results and analysis of BO along the Pareto front

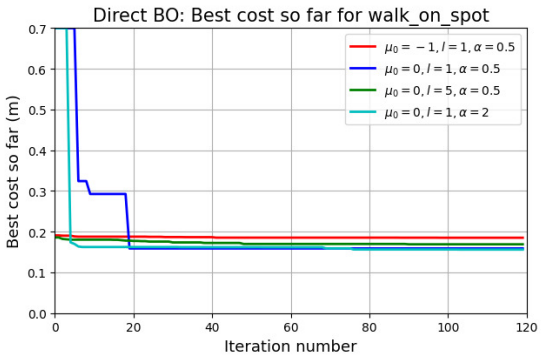
The final experiment of this thesis work is to run BO using only the NSGA-II-trained Pareto fronts as a search space. The main idea fueling this experiment is the desire to have the ability to train a Pareto front on a few trajectories, but then later have a means of adding new trajectories to the robot's repertoire without having to complete the MOO all over again. The Pareto front should ideally be diverse enough for a variety of types of new trajectories to succeed for at least one Pareto parameter set.

In addition to testing this approach on many different trajectories, we also perform trials with a similar set of hyperparameter variations as in the experiments of Section 4-2-2, C1-4. In order to gain a better understanding of how well this method performs, Figure 4-3 compares it to direct BO from the previous section, as well as the results of random search of the Pareto front space, averaged out over 1000 runs.

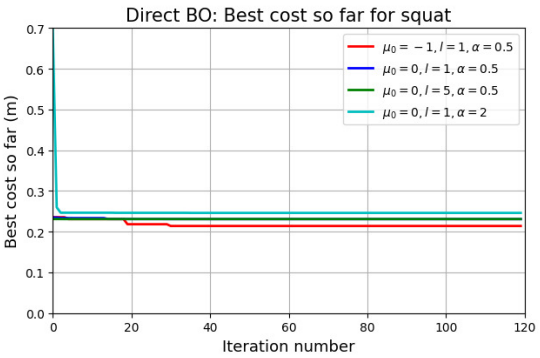
Figures 4-3 and 4-4 indicate that in most situations, this learning approach finds very good control solutions within 20 iterations. Additionally, it often finds non-failing solutions within the first few iterations. These characteristics suggest that this algorithm could also be applied on a real robot, not just in simulation.

Pareto-based BO vs. direct BO and Pareto-based random search

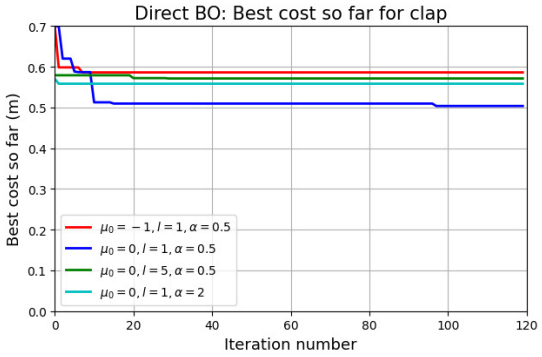
One immediate takeaway from Figure 4-3 is that direct BO finds conspicuously worse solutions than either the random search along the Pareto front or Pareto-based BO. This proves that there is a clear benefit to shrinking and discretizing the search space using prior information



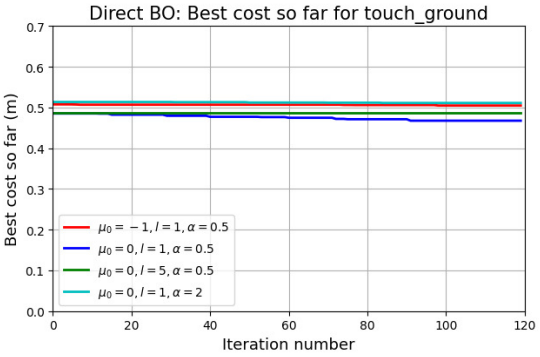
(a) Walk on spot.



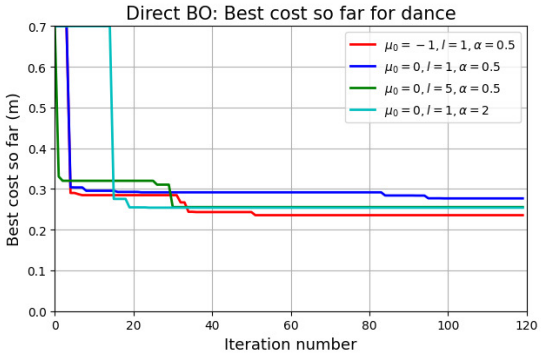
(b) Squat.



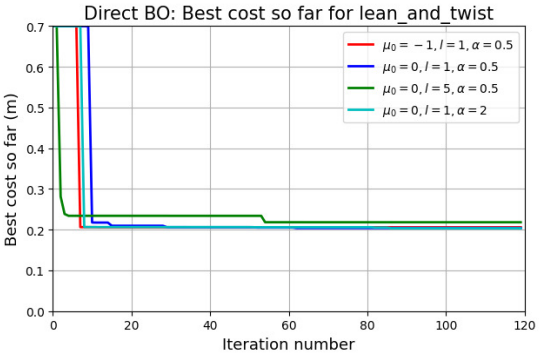
(c) Clap.



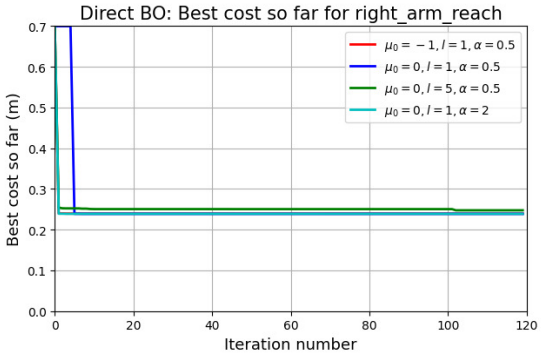
(d) Touch ground.



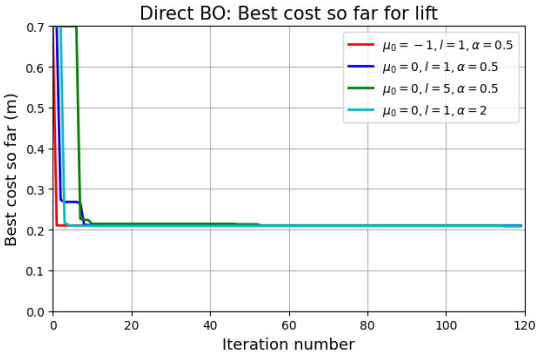
(e) Dance.



(f) Lean and twist.



(g) Right arm reach.



(h) Lift.

Figure 4-2: Direct BO comparison of performance on each trajectory of hyperparameter combinations from Table 4-1. A cost of 0.7 is used to represent all failing costs.

Trajectory	Pareto-based BO	Best Pareto-based BO combo
Walk on spot	0.1448	C3
Squat	0.2308	C4
Clap	0.4851	C1
Touch ground	1.0×10^{10} (fails)	N/A
Dance	0.1817	C2
Lean and twist	0.1768	C2
Right arm reach	0.1783	C4
Lift	0.1714	C3

Table 4-3: Best Cartesian task error achieved and its corresponding configuration for each training trajectory in Pareto-based BO.

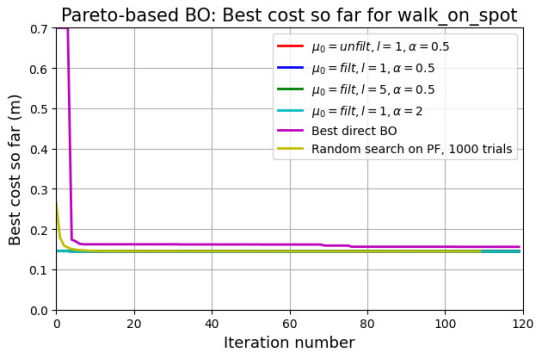
in the form of a Pareto front. The other obvious result is that for the touch ground trajectory, none of the Pareto-based methods find a single successful solution. This is a Set 2 trajectory, meaning it searched the Set 1 Pareto front for a solution. In fact, no successful solution exists for this trajectory on the Set 1 Pareto front, so in this particular case direct BO is the only viable approach. This points to either a lack of adequate diversity in the trajectory group which Set 1 is trained on, or an insufficient number of training generations with NSGA-II.

Comparison of different settings

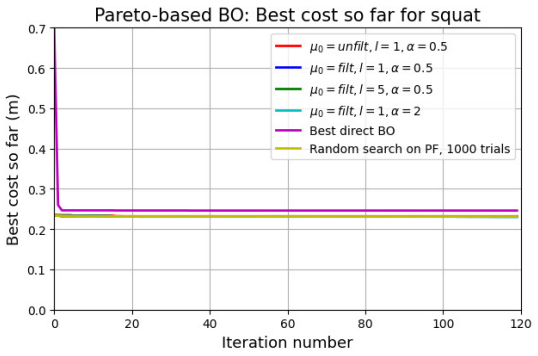
Unsurprisingly, every Pareto-based BO configuration performs much better than either direct BO or random search. Having access to either the filtered or unfiltered average of the Pareto fitnesses as an initial GP mean function effectively weeds out many of the parameter sets: sets that fail on most NSGA-II training trajectories are more likely to fail on a new trajectory. In order to examine the Pareto-based BO results more closely, zoomed-in plots are provided in Figure 4-4. Upon close inspection of the data in these zoomed-in plots, it appears that every configuration with filtered mean initialization (the optimistic option) performs almost identically on every trajectory. This likely means that the filtered mean already encourages exploration enough that increasing the α hyperparameter does not make much of a difference. The tiny differences that can be seen between C2 and C3 show that an increased smoothness hyperparameter does not perform quite as well. This means that within the Pareto front the solutions that perform well for a given trajectory are not all located in a single area of the Pareto front.

Notably, the filtered mean configurations outperform the unfiltered, pessimistic C1 for every trajectory except the clap (for which C1 performs better by only about 0.5 mm), as displayed in Table 4-3. In general, the filtered mean provides a better description of the search space compared to the unfiltered mean, which is full of extremely large error estimates.

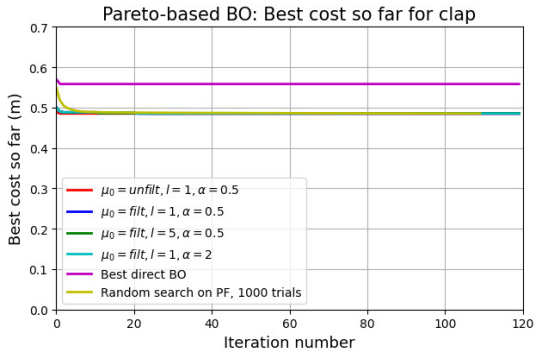
Finally, it is worth noting that for the dance trajectory, C1 does not find a single successful solution in 120 iterations. This means it performs even worse than random search and direct BO. A possible explanation for this behavior is that in the Set 1 Pareto front, out of 110 individuals, only 3 are non-failing for the dance trajectory. Since the C1 mean is initialized at very negative values for every individual (due to any one or more of the four train trajectories failing), it is not encouraged to explore and thus it gets stuck in the area of the front which has no successful parameter solutions.



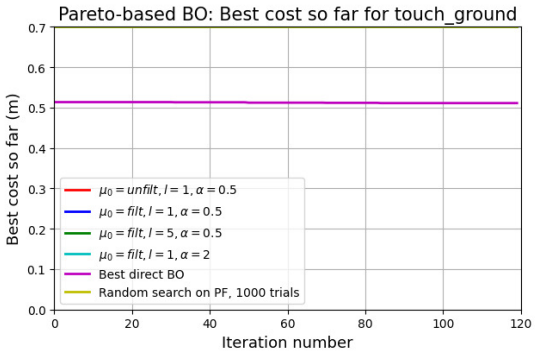
(a) Walk on spot.



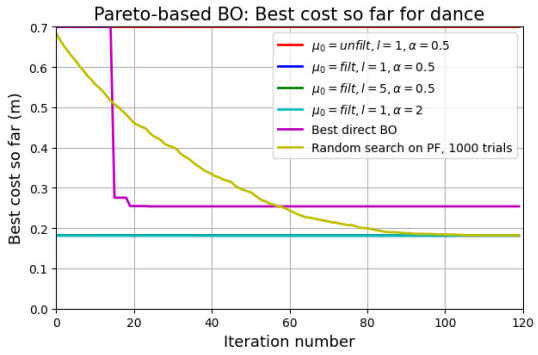
(b) Squat.



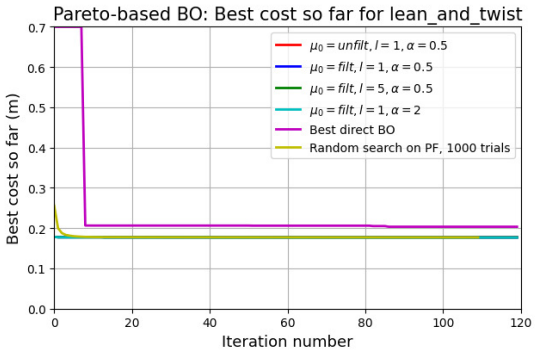
(c) Clap.



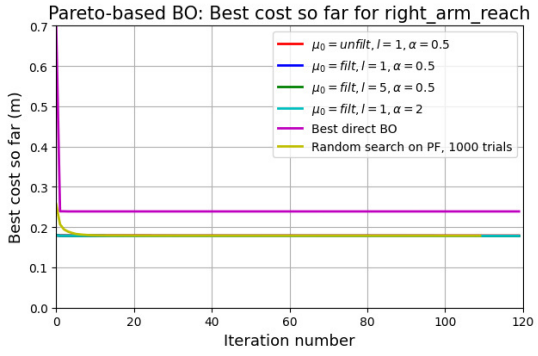
(d) Touch ground.



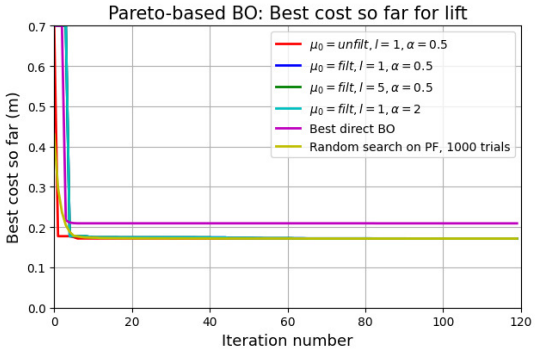
(e) Dance.



(f) Lean and twist.

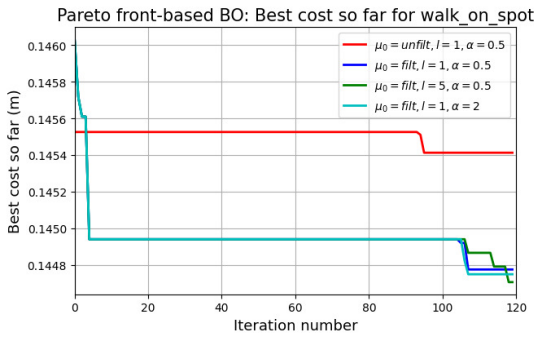


(g) Right arm reach.

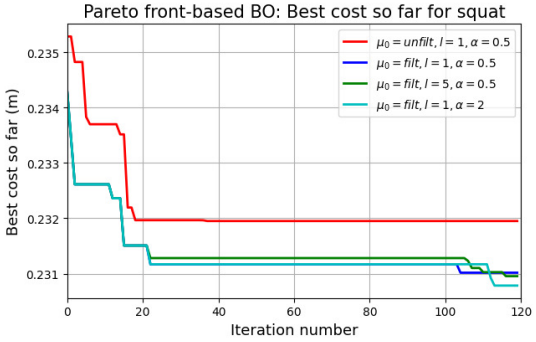


(h) Lift.

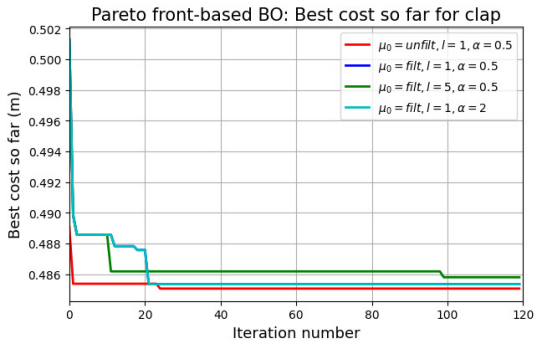
Figure 4-3: Pareto-based BO performance of hyperparameter combinations from Table 4-1, compared to direct BO and random search (along the Pareto front) results. A cost of 0.7 is used to represent all failing costs.



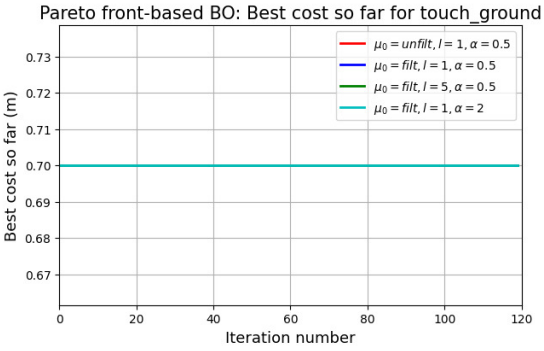
(a) Walk on spot.



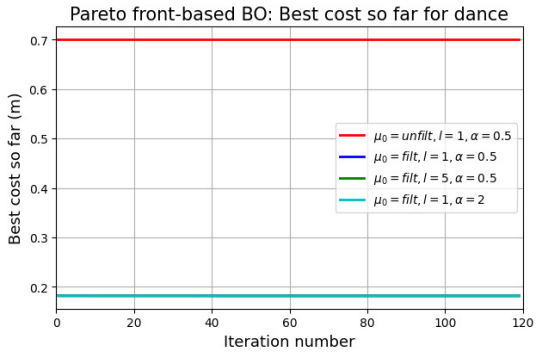
(b) Squat.



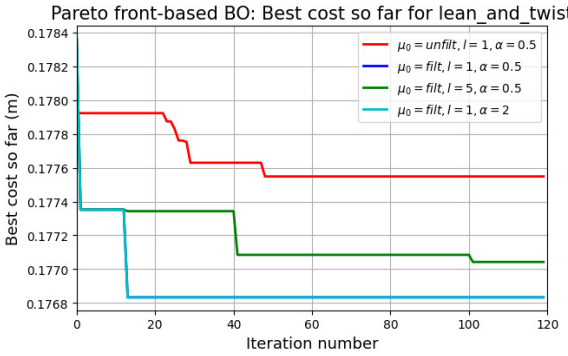
(c) Clap.



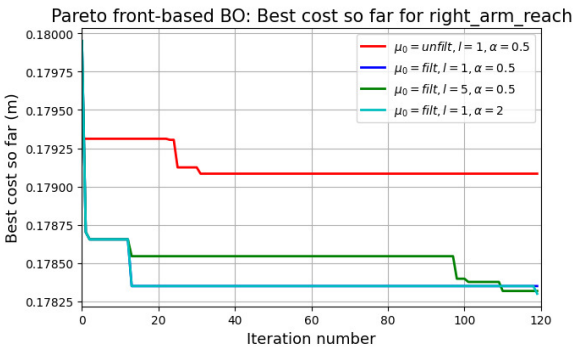
(d) Touch ground.



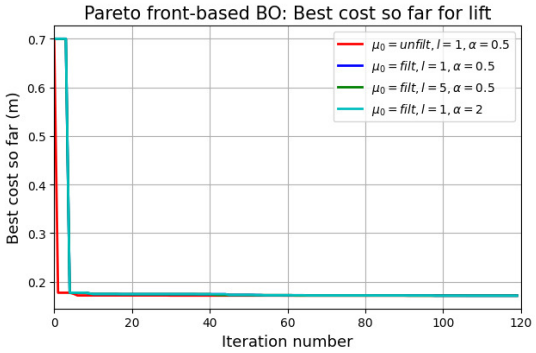
(e) Dance.



(f) Lean and twist.



(g) Right arm reach.



(h) Lift.

Figure 4-4: Zoomed in views of Pareto-based BO performance of hyperparameter combinations from Table 4-1. A cost of 0.7 is used to represent all failing costs.

4-3 Conclusion

The results of Pareto-based BO vastly outperform those of direct BO and random search along the Pareto front. This shows that providing a prior in the form of a parameter 'map' greatly speeds up learning and allows the algorithm to find better solutions. However, not all the results were positive: for the touch ground trajectory, no parameter set in the Pareto front was successful. This means that the quality of the pre-BO step, the generation of the Pareto front, should be improved. This can be done by perhaps adding a better variety of trajectories to the training set or by running the optimization for more generations in the hopes that better solutions are found over time.

Despite the lack of success on the touch ground trajectory, Pareto-based BO outperforms the other similar methods, can be tuned via the hyperparameters discussed previously in this chapter, and is overall successful in finding parameter solutions for new trajectories. Furthermore, because it learns acceptable solutions in very few trials, it could feasibly be applied on a real robot with a simulation-generated Pareto front, since the Pareto solutions are shown in Chapter 3 to transfer easily between different robot models.

Chapter 5

Discussion

This thesis work presents a solution for learning task-priority-based control parameters for a humanoid robot in two steps: optimizing a Pareto front of parameter sets with Non-dominated Sorting Genetic Algorithm II (NSGA-II), and searching in that Pareto front to learn suitable parameters for new trajectories with Bayesian optimization (BO). This combination of methods saves time that would otherwise be spent painstakingly hand-tuning controllers for various trajectories, and simultaneously improves the tracking accuracy of the robot on individual trajectories.

The experimental results in Chapter 3 show the superiority of this approach. Automatically tuning via NSGA-II yields a set of control solutions that contain not only better-performing 'robust' solutions than a hand-tuned controller, but also several solutions for each training trajectory which exhibit tracking performance far superior to that of the hand-tuned controller. Additionally, in this chapter it is found that the Pareto front retains a similar shape when tested on modified models, hence there is no need to use further optimization to adapt the Pareto front to new robot models.

In Chapter 4, the performance of BO along the NSGA-II-generated Pareto front is shown to be much better than that of direct BO and Pareto-based random search. Pareto-based BO successfully learns favorable control parameters for trajectories not part of the initial NSGA-II training set. In addition, this learning process finds a good solution in less than 20 trials, meaning that it also has the potential to be applied on the real robot.

5-1 Future work

The results of this thesis work are satisfying, but further development and testing should be done to ensure the design of the approach is optimal. First, the self-collision-checked NSGA-II optimization process should be run for a larger number of generations to flesh out the Pareto front. The optimization could also be run with a larger number of test trajectories in order to produce more diverse results. Furthermore, this work analyzes only a single Pareto dataset for each self-collision-checked training set. To ensure integrity of the analysis, several more tests

need to be done and the average results reported, such that parameter value estimates and performance on modified robot models are less likely to be influenced by potential outliers.

Another direction which deserves further investigation is to design and test other objective functions. The objective function used in this work is discontinuous: when the robot fails, regardless of what type of failure it is or what part of the trajectory at which it fails, the assigned cost is always the same. This means that there is a large area of the parameter search space where the fitness is constant, and thus no information is provided about which direction to go in to improve. It would be wise to attempt the optimization with an objective function that is more descriptive, such as one that accounts for how much of the trajectory the robot completes before failing, or something similar.

In terms of the BO step, it would be useful to test even more hyperparameter settings to fine-tune the performance of the algorithm. Specifically, it would be of interest to try a wider range of l_b and α values. Other ways of initializing the Gaussian process (GP) mean function could also be useful to test, such as by measuring which NSGA-II training trajectory is most similar to the BO trajectory via some heuristic, and then using that training trajectory's Pareto front performance as the initial mean for learning the new trajectory.

Most importantly, it is necessary to complete further tests on the real robot. More tests are needed to be certain that this approach for learning parameter sets actually does transfer well from simulation to reality.

Bibliography

[1] Hyginus, "Astronomica 2.1,"

[2] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. D. Prete, P. Soueres, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro, "TALOS: A new humanoid research platform targeted for industrial applications," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, IEEE, Nov 2017.

[3] E. Guizzo and E. Ackerman, "Darpa robotics challenge: A compilation of robots falling down," June 2015. [Online; posted 06-June-2015].

[4] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis, "OpenSoT: A whole-body control library for the compliant humanoid robot COMAN," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2015.

[5] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2019.

[6] L. Penco, E. M. Hoffman, V. Modugno, W. Gomes, J.-B. Mouret, and S. Ivaldi, "Learning robust task priorities and gains for control of redundant robots," *IEEE Robotics and Automation Letters*, vol. 5, pp. 2626–2633, Apr 2020.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, Apr 2002.

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

[9] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, pp. 328–347, Apr 2020.

[10] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, pp. 148–175, Jan 2016.

[11] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin: Springer, 2008.

[12] 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, pp. 1385–1400, Mar 2015.

[13] N. Hansen, D. V. Arnold, and A. Auger, "Evolution strategies," in *Springer Handbook of Computational Intelligence*, pp. 871–898, Springer Berlin Heidelberg, 2015.

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

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

[16] D. V. Arnold and N. Hansen, "A (1+1)-CMA-ES for constrained optimisation," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*, ACM Press, 2012.

[17] V. Modugno, G. Nava, D. Pucci, F. Nori, G. Oriolo, and S. Ivaldi, "Safe trajectory optimization for whole-body motion of humanoids," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, IEEE, Nov 2017.

[18] W. Gomes, V. Radhakrishnan, L. Penco, V. Modugno, J.-B. Mouret, and S. Ivaldi, "Humanoid whole-body movement optimization from retargeted human motions," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, IEEE, Oct 2019.

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

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

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. MIT Press Ltd, 2018.

[22] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Springer Tracts in Advanced Robotics*, pp. 9–67, Springer International Publishing, 2014.

[23] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, pp. 97–106, Mar 1964.

[24] J. Močkus, V. Tiesis, and A. Žilinskas, "The application of bayesian methods for seeking the extremum," *Toward Bayesian Optimization*, vol. 2, 1978.

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

[26] 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," 2010.

[27] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, pp. 5–23, Jun 2015.

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

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

[30] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, pp. 503–507, May 2015.

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

[32] A. D. Prete, N. Mansard, O. E. Ramos, O. Stasse, and F. Nori, "Implementing torque control with high-ratio gear boxes and without joint-torque sensors," *International Journal of Humanoid Robotics*, vol. 13, p. 1550044, Mar 2016.

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

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

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

[36] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 376–380, apr 1991.

Glossary

List of Acronyms

INRIA	National Institute for Research in Digital Science and Technology
LARSEN	Life-long Autonomy and interaction skills for Robots in a Sensing ENvironment
DoF	degrees of freedom
MOO	multi-objective optimization
BO	Bayesian optimization
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PS	policy search
SP	support polygon
ZMP	zero moment point
CoM	center of mass
WBC	whole-body control
SPW	soft priority weight
QP	quadratic programming
CG	convergence gain
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CCA	Constrained Covariance Adaptation
RBFs	radial basis functions
ProMPs	probabilistic movement primitives
RL	reinforcement learning
GP	Gaussian process
SE	Squared Exponential
PI	probability of improvement
EI	expected improvement

UCB	Upper Confidence Bound
NN	neural network
BBK	behavior-based kernel
KL	Kullback Leibler
DoG	Determinants of Gait
TSID	Task Space Inverse Dynamics

List of Symbols

α	Falling penalty
β	UCB tradeoff parameter
$\boldsymbol{\theta}_*$	New parameter set
$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Coriolis matrix
$\boldsymbol{e}$	Task Cartesian position error
$\boldsymbol{g}(\boldsymbol{q})$	Gravitational vector
$\boldsymbol{k}$	Kernel vector
$\boldsymbol{l}_k$	Length scale vector hyperparameter
$\boldsymbol{M}(\boldsymbol{q})$	Mass matrix
$\boldsymbol{p}_i$	Actual Cartesian task position
$\boldsymbol{q}$	Joint position vector
$\boldsymbol{R}$	Rotation matrix
$\boldsymbol{t}$	Translation vector
$\boldsymbol{x}_m$	Input point set
$\boldsymbol{x}_{m,transform}$	Transformed $\boldsymbol{x}_m$ point
$\boldsymbol{y}_m$	Output point set
$\boldsymbol{A}_i$	Task Jacobian
$\boldsymbol{b}_i$	Reference value
$\boldsymbol{e}$	Task error
$\boldsymbol{u}$	Control input
$\boldsymbol{u}_t$	Action
$\boldsymbol{x}_t$	State
$\ddot{\boldsymbol{q}}$	Joint acceleration vector
$\ddot{\boldsymbol{p}}_i^d$	Desired Cartesian acceleration
δ	GP-UCB hyperparameter
$\dot{\boldsymbol{A}}_i$	Time derivative of task Jacobian
$\dot{\boldsymbol{p}}_i$	Actual Cartesian task velocity
$\dot{\boldsymbol{q}}$	Joint velocity vector
$\dot{\boldsymbol{\omega}}_i^d$	Desired Cartesian angular velocity

$\dot{\boldsymbol{e}}$	Task velocity error
$\dot{\boldsymbol{p}}_i^d$	Desired Cartesian velocity
$\dot{\boldsymbol{q}}_i^d$	Desired joint velocity
ϵ	Regularization factor
$\hat{J}(\boldsymbol{\theta})$	Surrogate model of expected return
λ_i	Cartesian position gain
λ_i^D	Derivative convergence gain (CG)
λ_i^P	Proportional CG
$\mathcal{T}_i$	Task
$\mu(\boldsymbol{\theta})$	GP mean function
$\mu(\boldsymbol{\theta}_*)$	GP mean prediction
μ_i	Postural gain
Φ	Robot value
$\phi_{DoG}(\boldsymbol{\theta})$	DoG performance measure
ϕ_{NN}	NN performance measure for kernel
$\sigma^2(\boldsymbol{\theta}_*)$	GP variance prediction
σ_i	Cartesian orientation gain
σ_k^2	Signal variance hyperparameter
b	Trajectory index
c	Transformation scaling factor
d	Parameter vector dimension
$D_{1:t}$	Rollout data
$f(\boldsymbol{\theta})$	Objective function
i	Task index
K	Kernel matrix
$k(\boldsymbol{\theta}, \boldsymbol{\theta}')$	GP covariance function
k_{DoG}	DoG kernel
k_{trajNN}	Cost-agnostic neural network kernel
l	Number of objective functions
l_i	Hierarchy level selector
l_k	Length scale scalar hyperparameter
m	Umeyama point pair index
n	Number of point pairs for Umeyama's method
N_t	Total time steps
r_t	Reward
S	Feasible set
T	Final time
t	Current time
w_i	Soft priority weight
x^{cz}	Frontal zero moment point (ZMP) deviation from support polygon (SP) center

x_{SP}	Length of SP
y^{cz}	Horizontal ZMP deviation from SP center
y_{SP}	Width of SP