Task compatibility and feasibility maximization for whole-body control

Ryan Lober

To cite this version:

Ryan Lober. Task compatibility and feasibility maximization for whole-body control. Robotics [cs.RO]. Université Pierre et Marie Curie - Paris VI, 2017. English. NNT : 2017PA066597 . tel-01927038v2

HAL Id: tel-01927038
https://theses.hal.science/tel-01927038v2

Submitted on 19 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Pierre & Marie Curie
Institut des Systèmes Intelligents et de Robotique

PhD Dissertation

Task Compatibility and Feasibility Maximization for Whole-Body Control

Ryan Lober

Reviewers:
Ludovic Righetti
Olivier Stasse

Examiners:
Christopher Atkeson
Philippe Bidaud

Advisors:
Vincent Padois
Olivier Sigaud

November 10, 2017

Abstract

Highly redundant robots, such as humanoids, possess vast industrial and commercial potential. Unfortunately, producing useful behaviors on complex robots is a challenging undertaking, particularly when the robot must interact with the environment. Model-based whole-body control alleviates some of this difficulty by allowing complex whole-body motions to be broken up into multiple atomic tasks, which are performed simultaneously on the robot. However, tasks are generally planned without close consideration for the underlying controller and robot being used, or the other tasks being executed, resulting in infeasible and/or incompatible task combinations when executed on the robot. Consequently, there is no guarantee that the prescribed tasks will be accomplished, resulting in unpredictable, and most likely, unsafe whole-body motions.

The adverse side-effects of simultaneous task combinations have been well known to the robotics community since the inception of redundant robots. Typically, these effects are managed by prioritizing between tasks and tuning their gains and parameters, but never is their root cause eliminated. Planning techniques can account for additional tasks to improve whole-body motions prior to execution. Nevertheless, because of modeling errors, there are always differences between what is planned and what is executed on a real robot. On the other end of the spectrum, model-free learning methods attempt to bypass modeling errors by using reinforcement learning and policy search to incrementally improve task(s) through trial-and-error. Regrettably, these tasks must often be demonstrated kinesthetically to the robot beforehand, and require many trials to improve — this is no small feat on robots such as humanoids. Regardless of the technique used to generate the tasks, either the tasks themselves or their parameters need to be tuned on the real robot and this can be both time consuming and costly.

The objective of this work is to better understand what makes tasks infeasible or incompatible, and develop automatic methods of improving on these two issues so that the overall whole-body motions may be accomplished as planned. We start by building a concrete analytical formalism of what it means for tasks to be feasible with the control constraints and compatible with one another. By studying the underlying convex optimization problem produced by the model-based whole-body controller, we develop metrics for analyzing and quantifying these two phenomena. Using the model-based feasibility and compatibility metrics, we demonstrate how the tasks can be optimized using non-linear model predictive control, while also detailing the shortcomings of this model-based approach. In order to overcome these weaknesses, we then develop a model-free approach to quantify task feasibility and compatibility through simple controller and robot agnostic cost functions. Using these measures, an optimization loop is designed, which automatically improves task feasibility and compatibility using model-free policy search in conjunction with model-based whole-body control. Through a series of simulated and real-world experiments, we demonstrate the effects of infeasible and incompatible task sets, and show that by simply optimizing the tasks to improve both feasibility and compatibility, complex and useful whole-body motions can be realized. By endowing robots with an automated mechanism for correcting poorly designed tasks, we not only reduce the need for fine tuning of task priorities and parameters, but also open the door to more complex, robust, and useful whole-body behaviors.

Part I

FRONT MATTER

Contents

Abstract 3

I FRONT MATTER 5

List of Figures 13

List of Tables 21

List of Symbols 23

II INTRODUCTION 25

1 The Problem with Robots 27
- 1.1 How to Control a Robot 27
  - 1.1.1 Task Planning 28
  - 1.1.2 Task Servoing 29
  - 1.1.3 Whole-Body Control 30
- 1.2 The Problem with Modern Control Architectures 31
- 1.3 Related Work 32
  - 1.3.1 Manipulator Performance Measures 33
  - 1.3.2 Trajectory Optimization & Optimal Control 34
  - 1.3.3 Whole-Body Non-Linear Model Predictive Control 35
  - 1.3.4 Model-Free Policy Search 36
  - 1.3.5 Deep Learning Methods 38
- 1.4 Contributions 39
  - 1.4.1 List of Publications 41

2 Reactive Whole-Body Control 43
- 2.1 Free-Floating Rigid Body Dynamics 44
- 2.2 Control Objectives (Tasks) 44
  - 2.2.1 Acceleration Task 45
  - 2.2.2 Wrench Task 46
  - 2.2.3 Torque Task 46
- 2.3 Control Constraints 47
  - 2.3.1 Dynamics 47
  - 2.3.2 Actuator Limits 47
  - 2.3.3 Joint Limits 47
  - 2.3.4 Contact Constraints 48

2.4 Prioritization Strategies . . . 49
  2.4.1 Hierarchical Prioritization . . . 49
  2.4.2 Weighted Prioritization . . . 50
  2.4.3 Hybrid Schemes . . . 50
2.5 Task Servoing . . . 51
2.6 Conclusions . . . 51

III METHODS 53

3 Mathematical Foundations of Task Compatibility and Feasibility 55
3.1 Introduction . . . 55
3.2 A Brief Overview of Convex Optimization . . . 56
  3.2.1 A General Convex Problem . . . 56
  3.2.2 Quadratic Programming . . . 56
  3.2.3 Multi-Objective Optimization . . . 58
  3.2.4 Comparing Weighted Scalarization and Hierarchical Optimization . . . 58
3.3 Objective Compatibility and Feasibility . . . 60
  3.3.1 Objective Compatibility . . . 60
  3.3.2 Objective Feasibility . . . 66
3.4 Conclusions . . . 69

4 A Model-Based Approach to Maximizing Task Compatibility and Feasibility 71
4.1 Introduction . . . 71
4.2 Computing the Metrics . . . 72
  4.2.1 Equating Representations . . . 72
  4.2.2 Handling Equality Constraints . . . 73
  4.2.3 Computing the Compatibility and Feasibility Ellipsoids . . . 75
4.3 The Temporality of Compatibility and Feasibility . . . 75
4.4 Maximizing Task Compatibility and Feasibility . . . 75
4.5 Conclusions . . . 78

5 A Model-Free Approach to Maximizing Task Compatibility and Feasibility 81
5.1 Introduction . . . 81
5.2 Model-Free Formulation . . . 82
  5.2.1 Agent & Environment . . . 82
  5.2.2 States & Actions . . . 82
  5.2.3 Policies . . . 83
5.3 Policy Parameterizations . . . 83
  5.3.1 Learning From Demonstration . . . 84
  5.3.2 Waypoint Based Methods . . . 85
5.4 Policy Rollouts . . . 88
5.5 Task Compatibility and Feasibility Cost Functions . . . 89
  5.5.1 Tracking Cost . . . 89
  5.5.2 Goal Cost . . . 89
  5.5.3 Energy Cost . . . 90
5.6 Combining Cost Functions . . . 90
  5.6.1 Task Performance Cost . . . 90
  5.6.2 Total Performance Cost . . . 91
5.7 Task Compatibility and Feasibility Maximization (TCFM) . . . 92
  5.7.1 Determining Policy Parameters . . . 93

5.7.2 Update Strategy . . . 94
5.8 Conclusions . . . 96

IV RESULTS 97

6 Evaluation of Model-Based Metrics 99
6.1 General Experimental Setup . . . 99
6.2 Experimental Setup: Task Compatibility . . . 100
6.3 Results: Task Compatibility . . . 101
6.3.1 Strict Compatibility . . . 101
6.3.2 Compatibility Metrics . . . 102
6.4 Experimental Setup: Task Feasibility . . . 106
6.5 Results: Task Feasibility . . . 107
6.5.1 Feasibility Metrics . . . 107
6.6 Experimental Setup: Temporal Task Compatibility and Feasibility . . . 111
6.7 Results: Temporal Task Compatibility and Feasibility . . . 112
6.7.1 Compatibility Metrics . . . 113
6.7.2 Feasibility Metrics . . . 115
6.8 Conclusions . . . 116
6.8.1 Compatibility Metrics . . . 116
6.8.2 Feasibility Metrics . . . 117
6.8.3 Accounting for Null Spaces . . . 117
6.8.4 Most Useful Metrics . . . 117

7 Proof of TCFM Concept 119
7.1 Experimental Setup: Model-Free Costs vs. Model-Based Metrics . . . 119
7.2 Results: Model-Free Costs vs. Model-Based Metrics . . . 121
7.2.1 Model-Free Cost Analysis . . . 122
7.2.2 Comparing to Model-Based Metrics . . . 124
7.3 Experimental Setup: TCFM Validation . . . 124
7.3.1 Reparameterizing the Tasks . . . 125
7.3.2 TCFM Cost Function and Update Strategy . . . 127
7.4 Results: TCFM Validation . . . 127
7.4.1 Trajectory Tracking . . . 128
7.4.2 Cost Analysis . . . 129
7.4.3 Comparaison with Model-Based Metrics . . . 130
7.5 Conclusions . . . 130

8 Demonstrating TCFM on Complex Problems 131
8.1 Experimental Setup . . . 131
8.1.1 Test Case 1: Constrained Configuration . . . 133
8.1.2 Test Case 2: Workspace Violation . . . 133
8.2 Results . . . 134
8.2.1 Test Case 1: Constrained Configuration . . . 134
8.2.2 Test Case 2: Workspace Violation . . . 134
8.3 Conclusions . . . 137

9 Improving Sample Efficiency 139
9.1 Experimental Setup: GPTs as a Policy Parameterization . . . 139
9.1.1 Test Case 1: Constrained Configuration . . . 140
9.1.2 Test Case 2: Workspace Violation . . . 141
9.1.3 Test Case 3: Balance Perturbation . . . 141
9.2 Results: GPTs as a Policy Parameterization . . . 141
9.2.1 Test Case 1: Constrained Configuration . . . 142
9.2.2 Test Case 2: Workspace Violation . . . 143
9.2.3 Test Case 3: Balance Perturbation . . . 144
9.2.4 Conclusions: Effect of GPTs as a Policy Parameterization . . . 145
9.3 Experimental Setup: TCFM using GPTs and BO . . . 145
9.3.1 Test Case 1: Operational-Space Infeasibilities . . . 146
9.3.2 Test Case 2: Moving a Heavy Weight . . . 147
9.4 Results: TCFM using GPTs and BO . . . 148
9.4.1 Test Case 1: Operational-Space Infeasibilities . . . 148
9.4.2 Test Case 2: Moving a Heavy Weight . . . 150
9.4.3 Conclusions: TCFM using GPTs and BO . . . 151

10 Robust and Sample Efficient TCFM for Real Robots 155
10.1 Experimental Setup . . . 156
10.1.1 Test Case 1: Reaching . . . 156
10.1.2 Test Case 2: Standing . . . 158
10.1.3 Test Case 3: Standing with Help . . . 159
10.1.4 Test Case 4: Standing with a Different Whole-Body Controller . . . 159
10.2 Results . . . 161
10.2.1 Test Case 1: Reaching . . . 162
10.2.2 Test Case 2: Standing . . . 162
10.2.3 Test Case 3: Standing with Help . . . 163
10.2.4 Test Case 4: Standing with a Different Whole-Body Controller . . . 164
10.3 Conclusions . . . 167

V CONCLUSION 169

11 Conclusions and Perspectives 171
11.1 Is There Still a Problem with Robots? . . . 171
11.2 Limitations and Perspectives . . . 173
11.2.1 Evaluating the Metrics on a Humanoid . . . 173
11.2.2 Discrepancies Between κ^{NR} and κ_A^{NR} . . . 173
11.2.3 Postural Tasks are Probably a Bad Idea . . . 174
11.2.4 Automating Task Parameterization . . . 174
11.2.5 More Dimensions & Longer Problems . . . 175
11.2.6 Better Update Strategies . . . 175
11.2.7 Is Episodic Learning Closing the Loop? . . . 176
11.3 Future Work . . . 176
11.3.1 Implement the NMPC Problem . . . 176
11.3.2 TCFM Using Model-Based Metrics as the Cost Functions . . . 177
11.3.3 Reusing Surrogate Models Across Scenarios . . . 177
11.3.4 A Deep Model of Task Compatibility and Feasibility . . . 177
11.4 Closing Remarks . . . 177

VI BACK MATTER 179

A Appendix 181
A.1 Task and Constraint Formula References . . . 181
A.2 Objective Compatibility and Feasibility Metric References . . . 183
A.3 Model-Free Task Compatibility and Feasibility Cost Function References . . . 184
A.4 Dynamic Movement Primitives . . . 185
A.5 Gaussian Process . . . 186
A.6 Bayesian Optimization . . . 188

Index 189

Bibliography 191

List of Figures

1.1 A modern control hierarchy for highly redundant robotic systems, e.g. humanoid robots. At the lowest level is whole-body control, which determines the torques needed to accomplish a set of tasks. These tasks are controlled by the task servoing level where task trajectory errors are compensated using feedback. Finally the task trajectories are provided by high-level task planning, which is usually a combination of operator expertise and automated planning. 28

1.2 Incompatible and infeasible tasks which result in (a) the robot getting stuck in a constrained configuration and (b) a loss of equilibrium. 31

1.3 The task compatibility and feasibility maximization loop proposed in this paper is designed to correct incompatible and infeasible tasks produced by this modern control architectures. 32

1.4 Velocity ellipsoid described by Chiu [1987]. 33

1.5 Example of optimal task trajectories from [Neunert et al., 2017]. 35

1.6 Example of optimized multi-contact whole-body motion on the HRP-2 humanoid robot using whole-body NMPC from [Koenemann et al., 2015]. 36

1.7 Example of MFPS used to learn how to flip a pancake in [Kormushev et al., 2010]. 37

1.8 Example of learned operational-space wrench tasks from [Kalakrishnan et al., 2011]. . . . 37

1.9 The PR2 robot learning Deep MPC control for slicing food from [Lenz et al., 2015]. . . . 38

1.10 In (a), (b), and (c), a time-lapse is shown of an optimized standing motion performed by an iCub robot, which was found using the task compatibility and feasibility maximization algorithm developed in this work. The un-optimized motion caused the robot to fall. . . . 40

2.1 A diagram indicating visually what it means to include the root link pose in the parameterization. The 6-DoF of the floating base are modeled as a 6-DoF linkage with the world or inertial frame. Image taken from [Mistry et al., 2010]. 44

3.1 Two examples of convex functions. For each example, two test points, a and b, are chosen in the domain of $x \in \mathbb{R}^1$ to show that (3.4) is respected. This can be seen by observing that the chord which connects the points a and b lies on or within the function epigraphs, which are the sets of all points which lie above the function graphs. 57

3.2 Examples of unconstrained optima, $\boldsymbol{x}^*$, found using both weighted scalarization and hierarchical optimization. The individual unconstrained objective optima, $\boldsymbol{x}_i^*$, are shown as the blue dots. For the weighted case, (a), all weights are equal to 1.0. In the hierarchical case, (b), the numbering of the objective functions dictates the objective priority, with $i = 1$ being of highest priority. 59

3.3 Examples of constrained optima, $\boldsymbol{x}^*$, found using both weighted scalarization and hierarchical optimization. The individual unconstrained objective optima, $\boldsymbol{x}_i^* \; \forall i \in 1 \ldots 4$, are shown as the blue dots. The unconstrained optima found using both methods are shown in translucent red. 60

3.4 The compatibility ellipsoid, or Löwner John ellipsoid over the objective optima, $\boldsymbol{x}_i^*$, for the example problem from Sec. 3.2.4 65
3.5 (a) the feasibility ellipsoid for the example problem from Sec. 3.2.4. (b) evaluation of (3.63) for the example problem shows that only $\boldsymbol{x}_3^*$ lies within the feasibility ellipsoid and concordantly, inside the feasible set defined by the inequality constraints. 67
3.6 An example showing that even if the unconstrained objective optima lie within the feasibility ellipsoid, then there is still the possibility that their combination will result in an infeasible optimal value. Here, hierarchical optimization is used to produce $\boldsymbol{x}^*$, but this same result could be found with the right combinations of weights as well. 67
3.7 Scaled feasibility ellipsoid using $n = 2$ since $\boldsymbol{x} \in \mathbb{R}^2$. Evaluation of (3.63) shows now that both $\boldsymbol{x}_1^*$ and $\boldsymbol{x}_3^*$ lie within the scaled feasibility ellipsoid; however, $\boldsymbol{x}_1^*$ is outside of the true feasible set defined by the inequality constraints. 68
3.8 Evaluation of the two proposed objective feasibility measures. (a) shows the metric defined by Equations (3.67) and (3.68), and (b) shows the metric defined by (3.69). 69

5.1 In MFPS, the problem is composed of an agent (control architecture) which interacts with its environment (robot + surroundings) using actions, $\boldsymbol{a}(k)$, the environment's state, $\boldsymbol{s}(k+1)$, changes and a cost, $j(k)$ is associated with that interaction. 82
5.2 A visual description of the MFPS policy formulation and parameterization. The parameterized policy generation function is the collection of task trajectory generators. Their parameters are the policy parameters. The policies then contain the task servoing and whole-body control layers of the model-based control hierarchy. This policy determines the actions to take, (torques), given the current state of the environment. 83
5.3 An example of a 3D task trajectory with variance. This figure shows how variance can be computed given a Gaussian Process Trajectory, or GPT, then mapped to the weights of the individual DoF of the task. 87
5.4 A depiction of the TCFM policy search cycle. The critical components in this cycle are the cost function, update strategy, and policy parameterization scheme. This cycle is described in Algorithm 3. 92
5.5 A graphical representation of the CMA-ES policy update process. Here, the distribution of the parameter and cost data is used to randomly sample the solution space and the results are used to update the distribution in the direction of the natural gradient. Courtesy of `https://en.wikipedia.org/wiki/CMA-ES`. 95

6.1 The three task compatibility cases. The desired positions for the EE and elbow tasks are shown by the blue and red spheres respectively. For case 1, the starting positions are the same as the goal positions so they are not shown. For case 3, trajectories are used and shown by the blue and red lines. 101
6.2 (a) rank difference for the linear system of equations defined by the task objective functions. Only cases 1 and 3 are initially strictly compatible, but both loose strict compatibility after the first few time steps do to changes in the desired accelerations. (b) desired acceleration norms for each task for each case. (c) zoom on the desired acceleration norms for task in case 1. 102
6.3 Nuclear norm ratios κ^{NR} and κ_A^{NR} calculated for each case using the standard task-only formulation (see (4.11)) and the augmented equations of motion formulation (see (4.13)), respectively. 103
6.4 Computation of the task compatibility metrics for the three compatibility cases studied here. (a) shows the Optimum Distance (OD) metric. (b) shows the Optimum Cost (OC) metric. (c) shows the Ellipsoid Distance (ED) metric. 104

6.5 These figures show the variations of the optimal values for the elbow position task, $\boldsymbol{\chi}^*_{\mathrm{EL}}$, and the prioritized optimum, $\boldsymbol{\chi}^*$, for all three cases and demonstrate that the displacement of $\boldsymbol{\chi}^*_{\mathrm{EL}}$ modifies the κ^{OD} metrics shown in Fig. 6.4a, not the changes in $\boldsymbol{\chi}^*$. 104

6.6 Shows the distance between the compatibility ellipsoid center and the prioritized optimum and the Ellipsoid Volume (EV) compatibility metrics. The computation of the compatibility ellipsoid is highly sensitive to the sparsity of this problem. 105

6.7 The robot motions for the three task feasibility cases. In case 1, the robot maintains its initial feasible posture. In case 2, the reference posture is set immediately to $\boldsymbol{q}_{\mathrm{IF}}$, an infeasible posture, and the robot moves quickly to the reference. In case 3, a joint-space task trajectory is used to move the robot more gradually to $\boldsymbol{q}_{\mathrm{IF}}$. 107

6.8 Evolution of the joint torques and positions for each of the three feasibility case studies. The upper and lower bounds are indicated by the darkened black horizontal lines near the y-axis upper and lower limits on each graph. 108

6.9 (a) Optima Distance, ϕ^{OD}, and Ellipsoid Distance, ϕ^{ED}, metrics for cases 1-3 of the task feasibility experiment. Because there is only one task in this experiment, $\phi^{\mathrm{OD}} = \phi^{\mathrm{ED}}$. (b) Ellipsoid Evaluation metric, ϕ^{EE}, with and without scaling for cases 1-3 of the task feasibility experiment. The black horizontal line on the upper graph indicates the value of 1.0. 108

6.10 Torque limit lower bound inequality constraint on the q_2 articulation for case 2. 109

6.11 The evolution of the inequality constraints for the lower joint position limits for case 3, evaluated at the prioritized optimum, $\boldsymbol{\chi}^*$. The $G\boldsymbol{\chi}^*$ line is darkened where the prioritized optimum would violate the constraint. A violation indicates that the prioritized optimum is infeasible and thus activates certain constraints in the QP. 110

6.12 (top) the Ellipsoid Volume metric, ϕ^{EV} for the task feasibility experiment. (middle) the norm of the feasibility ellipsoid center. (bottom) the norm of the postural task optimum. 110

6.13 Time-lapse of the motion used to explore the temporality of task compatibility and feasibility. 111

6.14 Task reference and real position values throughout the movement. As can be seen, the trajectory tracking suffers in the middle of the movement, which indicates some decrease in task compatibility/feasibility. 112

6.15 The evolution of the joint torques and positions for the temporal example. During the movement, q_1 and q_3 hit their lower position limits, thus indicating an infeasibility. 112

6.16 Top: κ^{NR} calculated using the standard task-only formulation (see (4.11)). Bottom: κ^{NR}_A calculated using the augmented equations of motion formulation (see (4.12)). 113

6.17 The desired acceleration norms for each task. 113

6.18 (a) Optimum Distance metrics, κ^{OD}, for the temporal example. (b) (top) the Euclidean distance between $\boldsymbol{\chi}^*$ and the ellipsoid center, $\boldsymbol{c}_\kappa$, and (bottom) the norm of $\boldsymbol{\chi}^*$. (c) Optimum Cost metrics, κ^{OC}. 114

6.19 (a) Optima Distance metric, ϕ^{OD}, and (b) Ellipsoid Distance metric, ϕ^{ED}, for the temporal experiment. 115

6.20 (a) shows the Ellipsoid Evaluation metrics for each of the tasks (top) and for the prioritized optimum, $\boldsymbol{\chi}^*$, (bottom). The dashed horizontal black line indicates a value of 1.0. (b) shows the evolution of the inequality constraints for the lower joint position limit of articulation q_3 at $\boldsymbol{\chi}^*$ (top), and the distance between the prioritized optimum and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$ (bottom). Zero distance indicates that the prioritized optimum is completely feasible with the given constraints. Any positive distance indicates that the prioritized optimum lies outside of the feasible set and cannot be perfectly attained. 116

7.1 Time-lapse of the motion produced by the case 2 policy. 120

7.2 Real and reference values for the EE and elbow task trajectories for the case 1 and case 2 rollouts. The case 1 trajectories are in green and the case 2 trajectories are in dashed red. The reference positions are indicated by the lighter lines and the real positions by the darker lines. . . . 121
7.3 Time evolution of the tracking, $j_i^{\mathrm{TP}}(k)$, goal, $j_i^{\mathrm{TP}}(k)$, and task performance, $j_i^{\mathrm{TP}}(k)$, costs for each task in the example problem from Sec. 6.6. The costs are calculated as the area under the curves, but looking at their evolution allows the task performance to be analyzed. 122
7.4 Evolution of the total task performance cost, $j^{\mathrm{TP}}(k)$ (top), energy cost, $j^{\mathrm{E}}(k)$ (middle), and total performance cost, $j^{\mathrm{P}}(k)$ (bottom), for cases 1 and 2. . . . 123
7.5 Model-based task compatibility and feasibility metrics from Chapter 3. . . . 124
7.6 The model-free policy and its parameterization for the example problem studied here. This figure is adapted from Fig. 5.2 . . . 125
7.7 Example of the reparameterized task. This graph shows the x dimension of the EE task trajectory. . . . 126
7.8 The reparameterized policy function uses clamped splines to generate the trajectory and allows "adjustable" parameters (waypoints) to be added to the EE and elbow tasks. These parameters permit TCFM to optimize the tasks. The joint position task parameters, $\boldsymbol{\theta}_{\mathrm{JP}}$, are shown to indicate that they are remain a part of the policy but since they are constant they are no longer part of the policy parameterization, $\boldsymbol{\theta}$. . . . 126
7.9 Time-lapse of the original and TCFM optimized policies. As can be seen from the images, the optimized policy better tracks its reference trajectories (the blue and red lines) than the original policy and arrives at its goal positions in 6 seconds rather than the original 8 seconds. . . . 127
7.10 Real and reference values for the EE and elbow task trajectories for the original and optimized policies. The original task trajectories are in green and the optimized task trajectories are in dashed blue. The reference positions are indicated by the lighter lines and the real positions by the darker lines. . . . 128
7.11 Time evolution of the tracking, $j_i^{\mathrm{TP}}(k)$, goal, $j_i^{\mathrm{TP}}(k)$, and task performance, $j_i^{\mathrm{TP}}(k)$, costs for each task in the example problem from Sec. 6.6 is used. The costs are calculated as the area under the curves, but looking at their evolution allows the task performance to be analyzed. . . . 128
7.12 Evolution of the total task performance cost, $j^{\mathrm{TP}}(k)$ (top), energy cost, $j^{\mathrm{E}}(k)$ (middle), and total performance cost, $j^{\mathrm{P}}(k)$ (bottom), for original and optimized policies. . . . 129
7.13 (left) scaled performance cost curve for the TCFM in the proof of concept example. (right) the same curve but plotted by its component scaled costs. . . . 129
7.14 Model-based task compatibility and feasibility metrics from the original and optimized policies. . . . 130
7.15 Distance between the prioritized optimum, $\boldsymbol{\chi}^*$, and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$, for the original and optimized policies. Any non-zero distance means the prioritized optimum violates the inequality constraints. . . . 130

8.1 (a) - (d) show the constrained configuration scenario and (e) - (h) show the workspace violation scenario. (a) & (e) starting positions. (b) & (f) left hand + standing tasks end configurations. (c) & (g) right hand + standing tasks end configurations. (d) & (h) end configurations of left hand, right hand and standing task combinations where the robot cannot attain both goal positions (spheres) simultaneously. . . . 132
8.2 Evolution of torso DoFs during the execution of the left hand + standing and right hand + standing combinations. . . . 134
8.3 Cost curves for the TCFM optimizations for both test cases. Note that each update corresponds to 15 policy rollouts. For all cases, the TCFM, using DMPs and $\mathrm{PI}^{\mathrm{BB}}$, is stopped after 300 rollouts, because the update policy convergence criterion is never attained. 135

8.4 Optimized task combinations found with TCFM. (a) - (d) show a time-lapse of the optimized tasks for test case 1 from Sec.8.1.1. 135
8.5 A plot of the distances between the hand task trajectories for each instant in time when they are simultaneously active. The optimized DMPs without task deactivation delay this incompatibility, and the optimized DMPs with task deactivation remove it. 136
8.6 Optimized task combinations found with TCFM. (a) - (h) show two time-lapses of the the optimized tasks for test case 2 described in Sec. 8.1.2. (a) - (d) show the solution found without task deactivation and (e) - (h) show the solution found with task deactivation. . . 136

9.1 The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters. 142
9.2 The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters. 143
9.3 The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters. 144
9.4 This figure shows the original and optimized policies for the "Operational-Space Infeasibilities" test case detailed in Sec. 9.3.1. (a) shows the initial starting posture. In (b) it can be seen that the robot is blocked by the yellow obstacle introduced at 1.0s. TCFM generates a trajectory which causes greater acceleration at the beginning of the movement, moving the right arm above the obstacle before it arrives (c). This allows the robot to reach its goal, indicated by the red sphere. 148
9.5 (a) and (b) Plots of the z component of the original and optimized hand trajectories, using static and variable weights respectively. The notation $\bullet^*$ indicates an optimal value, or a value associated with the optimized policy. (c) and (d) a plot of the surrogate function cost mean, $\hat{j}^P_\mu$, of the BO update strategy within the TCFM, using static and variable weights respectively. These figures show the learned performance cost landscape of the obstacle avoidance simulation, after convergence of TCFM. The task rollouts, or cost function samples, are plotted by the green dots. The blue dot represents the optimal variables found for this simulation. Using these values for the task's optimization waypoint, the obstacle is successfully avoided, i.e. the effects of the task incompatibilities and infeasibilities are removed. 149
9.6 This figure shows the original and optimized tasks for the "Moving a Heavy Weight" simulation detailed in Sec. 9.3.2. (a) shows the initial starting posture. In (c), it can be seen that the robot comes up just short of its goal location. By optimizing the middle waypoint of the task, the robot is able to accelerate more at the beginning of the movement, building up sufficient inertia to reach its goal within the acceptable error, as demonstrated by (d). 150
9.7 Evolution of the performance cost in percentage using static, (a), and variable, (b) weights (deterministic and stochastic policies). Each rollout cost is broken up into the component tracking, j^{T}, goal, j^{G}, and energy, j^{E}, costs. 151
9.8 A comparison of the TCFM with deterministic and stochastic policies (static and variable weights). 152

10.1 Figures (a) and (b) show the policy parameter, $\boldsymbol{\theta}$, bounding boxes used by the TCFM update strategies for the reaching and standing scenarios, respectively. 156
10.2 Random reaching targets used to provide a statistical analysis of TCFM. The target spheres are color coded to indicate their test case, with green meaning reachable, orange meaning possibly reachable, and red meaning unreachable. 157

10.3 Results of 100 reaching experiments used to study the task compatibility optimization method. An average example is presented for the three possible reach cases. In the reachable case, (a) & (b), both the original and optimized policies attain the reach target. In the possibly reachable case, (d) & (e), the original policy does not attain the target but the optimized policy does. Finally in the unreachable case, (g) & (h), neither policy attains the target, but the optimized policy reduces the target error. For each case, the scaled cost means and standard deviations are plotted for both the BO and CMA-ES update strategies. Any scaled cost lower than 1.0 is an improvement (the 1.0 line is indicated by a dashed grey lines). For all three cases the scaled performance cost $\hat{\jmath}^{\mathrm{P}^*}$ is always less than or equal to 1.0. This indicates that the optimized policies will always be as good if not better than the original policies. In the reachable case, (c), little improvement is seen because the tasks are already compatible. For the possibly reachable and unreachable cases, (f) and (i), the compatibility is improved by reducing the tracking and goal costs at the expense of increased energy usage. For each of the cases, BO outperformed CMA-ES in number of rollouts for convergence on average, but was less consistent across-the-board. 161

10.4 Original and optimized reaching behaviors executed on an iCub robot. These preliminary results show that the behaviors produced by TCFM are viable on real platforms. 162

10.5 Original and optimized CoM reference trajectories and their resultant whole-body motions. The original policy produces an unstable standing motion causing the robot to lose balance. The optimized policy, however, produces a successful sit-to-stand transition. The right hip is translucent in (b) to make the reference trajectory visible. (c) evolution of the CoM for the original and optimized policies. The original CoM curves are cut off after 2.7 seconds when the robot loses balance. The red dashed line indicates the moment when the bench contacts are deactivated in the whole-body controller. 163

10.6 Original and optimized CoM reference trajectories with and without support. (a) The original policy produces an unstable standing behavior causing the robot to lose balance. The optimized without support, (b), original policy with support, (c), and externally supported, (d), cases produce successful sit-to-stand transitions. (e) The costs presented here corresponds to the three following cases: optimized policy without external support, original policy with external support and optimized policy with external support. Each cost is presented as a ratio of the original cost (no optimization, no external support). As intuitively expected, the use of an external supporting force together with an optimized policy yields the best results in terms of tracking and whole-body energy expenditure. (f) Evolution of the CoM for the original and optimized policies, with and without support. The original CoM curves without support are cut off after 2.7 seconds when the robot loses balance. The red dashed line indicates the moment when the bench contacts are deactivated in the whole-body controller. 164

10.7 Evolution of the CoM trajectories of the original and optimized policies. "B" indicates the bootstrapped case, and "NB" the non-bootstrapped case. B 0 is the original policy executed in simulation. The solid lines are the reference values generated by $\boldsymbol{\pi_\theta}$ and the lighter dashed lines are the real measured values. The original, B 0, real lines are cut off after 2.5s when the robot falls. The noisy B 0 force profile is omitted from the force plot, to not obfuscate the other force profiles. 165

10.8 (a) performance cost percentages (bootstrapped case) from the rollouts in both simulation and on the real robot. The rollouts which produced a failure (falling) are indicated by the red hatched backgrounds. The optimal (best observed costs) policy parameters, $\boldsymbol{\theta}^*$, are indicated for both real rollout cases. (b) costs for the non-bootstrapped case. The optimal policy found in the simulated rollouts comes from B 21 and is indicated by the yellow star. B 25 and NB 0, i.e the first real rollouts for the bootstapped and non-bootstrapped cases, use the B 21 policy and are also indicated by yellow stars. B 33 is the optimal policy found during the real bootstrapped rollouts. NB 2 is the optimal policy found during the real non-bootstrapped rollouts. (c) initial posture of the iCub robot. (d) and (e) final standing postures of the optimized motions for the bootstrapped and non-bootstrapped cases respectively. 166

10.9 Evolution of the torso pitch joint torques for the rollouts 25 and 33 in the bootstrapped case. 167

11.1 The task compatibility and feasibility maximization loop proposed in this dissertation is designed to correct incompatible and infeasible tasks produced by this modern control architectures. This loop is one of trial-and-error learning, thus, the control frequency is replaced by the number of rollouts. 172

List of Tables

3.1 Computation of the objective compatibility measure defined in (3.47) for the $\boldsymbol{x}^*$ calculated using weighted scalarization and hierarchical optimization — both in the unconstrained case. 64

4.1 Compatibility metrics formulated using the whole-body controller variables from Chapter 2. 72
4.2 Feasibility metrics formulated using the whole-body controller variables from Chapter 2. . 73

6.1 The joint position limits and some specific configurations of the PUMA 560 robot simulation used here. The variable $\boldsymbol{q}_N$ is the nominal posture for starting the robot, $\boldsymbol{q}_V$ is the vertical posture which puts each articulation in the middle of its joint range defined by $\boldsymbol{q}_{\min}$ and $\boldsymbol{q}_{\max}$, and $\boldsymbol{q}_{IF}$ is an infeasible joint posture which is $\boldsymbol{q}_{\min} - 10°$. 100
6.2 Task set for the task compatibility experiments. $K_d = 2\sqrt{K_p}$ for all tasks. 100
6.3 Task set for the task feasibility experiments. $K_d = 2\sqrt{K_p}$ for all tasks. 106
6.4 Constraints for task feasibility experiments. 106
6.5 Constraints for the temporal task compatibility and feasibility experiments. 111

7.1 Task set. $K_d = 2\sqrt{K_p}$ for all tasks. 119
7.2 Constraints. 120
7.3 Task trajectory durations in seconds for cases 1 and 2. 121
7.4 The sums of the tracking, goal and task performance costs for cases 1 and 2. 122
7.5 Task set for the TCFM proof of concept problem. $K_d = 2\sqrt{K_p}$ for all tasks. 124
7.6 Constraints for the TCFM proof of concept problem. 125

8.1 Task set for test cases 1 and 2 of TCFM with DMPs and $\mathrm{PI}^{\mathrm{BB}}$. $K_d = 2\sqrt{K_p}$ for all tasks. 132
8.2 Constraints for test cases 1 and 2 of TCFM with DMPs and $\mathrm{PI}^{\mathrm{BB}}$. 132

9.1 Task set for test cases 1 of the GPTs as a policy parameterization experiment. $K_d = 2\sqrt{K_p}$ for all tasks. 140
9.2 Constraints for test cases 1 of the GPTs as a policy parameterization experiment 140
9.3 Task set for test cases 3 of the GPTs as a policy parameterization experiment. $K_d = 2\sqrt{K_p}$ for all tasks. 141
9.4 Constraints for test cases 3 of the GPTs as a policy parameterization experiment 141
9.5 Task set for test case 1 of TCFM with GPTs and BO. $K_d = 2\sqrt{K_p}$ for all tasks. 146
9.6 Task set for test case 2 of TCFM with GPTs and BO. $K_d = 2\sqrt{K_p}$ for all tasks. 147
9.7 Constraints for test case 2 of TCFM with GPTs and BO. 147

10.1 Task set for test case 1 of TCFM with TOTs and CMA-ES/BO. $K_d = 2\sqrt{K_p}$ for all tasks. 157
10.2 Constraints for test case 1 of TCFM with TOTs and CMA-ES/BO. 157
10.3 Task set for test case 2 of TCFM with TOTs and CMA-ES/BO. $K_d = 2\sqrt{K_p}$ for all tasks. 158
10.4 Constraints for test case 2 of TCFM with TOTs and CMA-ES/BO. 158

10.5 Task set for test case 4 of TCFM with TOTs and CMA-ES/BO. Levels are indicated in ascending order of importance. 159
10.6 Constraints for test case 4 of TCFM with TOTs and CMA-ES/BO. 160

A.1 A summary of the different task formulations for the whole-body controller. The original equation references for each task are indicated in the "reference" column. 181
A.2 A summary of the different constraint formulations for the whole-body controller. The original equation references for each constraint are indicated in the "reference" column. . . 182
A.3 Objective compatibility metric summary. The variable n_o is the number of objective functions, n the dimension of the optimization variable, $\boldsymbol{x}$, $\boldsymbol{c}_\kappa$ the compatibility ellipsoid center, B_κ, the compatibility ellipsoid matrix, and $\boldsymbol{x}^*$ is the overall global unconstrained problem optimum. 183
A.4 Objective feasibility metric summary. The variable n_o is the number of objective functions, n_c is the number of constraint equations, n the dimension of the optimization variable, $\boldsymbol{x}$, $\boldsymbol{c}_\kappa$ and $\boldsymbol{c}_\phi$, the compatibility and feasibility ellipsoid centers, P_ϕ and B_ϕ, the feasibility ellipsoid matrices, and $\boldsymbol{x}^*$ is the overall global unconstrained problem optimum. 183
A.5 A summary of the various cost functions used to evaluate policies. These functions are designed to capture various aspects of whole-body control performance degradation resulting from task incompatibilities and infeasibilities. The term k indicates the control time step, and N the total number of time steps. The actual total duration of the rollout is defined as $t_{\boldsymbol{\pi}}^{\text{end}} = Nh$, where h is the control sampling period. The term $\boldsymbol{\epsilon}(k)$ is the task position tracking error at time k, and $\boldsymbol{\epsilon}^{\text{goal}}(k)$, is the difference between the current task position and its last waypoint $\boldsymbol{\lambda}_{n_\lambda}$ at the k^{th} time step. 184

List of Symbols

A description of several symbols used in this document document.

β	task variance to weight scaling factor
η	$\mathrm{PI}^{\mathrm{BB}}$eliteness factor
j^{E}	Energy Cost
j^{G}	Goal Cost
j^{P}	Performance Cost
$\boldsymbol{\nu}$	Derivative of the generalized coordinates with the floating-base.
j^{TP}	Task Performance Cost
j^{T}	Tracking Cost
κ	objective compatibility metric
κ^{ED}	Ellipsoid Distance (ED) compatibility metric
κ^{EV}	Ellipsoid Volume (EV) compatibility metric
κ^{NR}	Nuclear norm Ratio (NR) compatibility metric
κ^{OC}	Optimum Cost (OC) compatibility metric
κ^{OD}	Optimum Distance (OD) compatibility metric
ϕ	objective feasibility metric
ϕ^{ED}	Ellipsoid Distance (ED) feasibility metric
ϕ^{EE}	Ellipsoid Evaluation (EE) feasibility metric
ϕ^{EE^*}	Ellipsoid Evaluation (EE*) feasibility metric
ϕ^{EV}	Ellipsoid Volume (EV) feasibility metric
ϕ^{OD}	Optima Distance (OD) feasibility metric
$\boldsymbol{q}$	Generalized coordinate parameterization.
φ	energy cost scaling factor
CMA-ES	Covariance Matrix Adaptation - Evolutionary Strategy
DoF	Degree(s) of Freedom

PI^{BB}	Policy Improvement though Black-Box optimization
TCFM	Task Compatibility and Feasibility Maximization
w.r.t.	with respect to
BO	Bayesian Optimization
CoM	Center of Mass
CoP	Center of Pressure
DDPG	Deep Deterministic Policy Gradient
DMP	Dynamic Movement Primitive
DRL	Deep Reinforcement Learning
GP	Gaussian Process
GPT	Gaussian Process Trajectory
ID	Inverse Dynamics
IVK	Inverse Velocity Kinematics
LIP	Linear Inverted Pendulum
LQG	Linear Quadratic Gaussian
LTI	Linear Time-Invariant
MFPS	Policy Search
MPC	Model Predictive Control
NMPC	Non-linear Model Predictive Control
PID	Proportional-Integral-Derivative
PoS	Polygon of Support
QP	Quadratic Program
RRT	Rapidly exploring Random Tree
s.t.	subject to
SDP	Semidefinite Programming
SLQ	Sequential Linear Quadratic
SVD	Singular Value Decomposition
TOPP	Time-Optimal Path Parameterization
TOT	Time-Optimal Trajectory
ZMP	Zero Moment Point

Part II

INTRODUCTION

Chapter 1

The Problem with Robots

When controlling complex robots like humanoids, it is often the case that the executed behavior does not match that which was planned, and the ability to modify the behavior online is a crucial missing piece in the architecture of modern control schemes. In this chapter, we explain why this mismatch between planning and execution occurs, and show how a variety of methods have been used to solve this problem. Drawing from these works, we propose methods for analyzing and correcting this mismatch. At the end of the chapter, the main contributions of this work are summarized and an overview of the document layout is presented.

Complex robots, such as humanoids, can execute multiple simultaneous tasks and perform complicated whole-body behaviors. However, the problem with these robots is that when they perform multiple simultaneous tasks, they often do not execute them exactly as planned. The result is typically a failure to accomplish the desired behavior, and in the case of humanoids, usually a costly repair as well. This happens because the models used to plan and execute the tasks are always somewhat inaccurate, do not account for the other tasks being executed, and generally, ignore the robot's constraints. Planning in this way produces tasks which may be incompatible with one another, infeasible with the control constraints, and ultimately impossible to properly execute. Unfortunately, re-planning the tasks is generally not a good option, because it is either too slow to be done on-line, or the source of the task incompatibilities and infeasibilities in the first place. The objective of this work is to introduce the necessary tools and techniques to understand mathematically what it means for tasks to be incompatible and infeasible, measure how incompatible and infeasible they are, and optimize them to maximize their compatibility and feasibility, without re-planning. In order to understand exactly why this is a problem, it is helpful to look at how complex robots are controlled.

1.1 How to Control a Robot

When using the term robots in this work, we are referring to redundant robots, and specifically, *humanoids*, unless otherwise stated. However, despite being geared towards humanoids, the work presented here is just as applicable to any robot. The control of robots begins with high-level goals such as "reach for that object while balancing". Complex goals like these must be translated into tasks for the robot, which are operational-space and joint-space objectives for all or part of the robotic mechanism. The process of converting high-level goals to tasks is termed *task planning*. This high-level task planning may be automated or done by an expert operator, and generally takes seconds to minutes ($\leq$ 0.1Hz), making it open-loop. The output is one or more tasks and their associated trajectories, which are needed to accomplish high-level goals. A *whole-body controller* is then used to determine the torques to send to the robot actuators at each control time step ($\sim$ 1kHz) to minimize the error between the actual task values, and the reference task values provided by the planned task trajectories. Unfortunately, this control scheme is typically subject to a large degree of error and drift in the task trajectory following. To avoid task divergence, a layer of *task servoing* is placed between whole-body control and the task planning. Task servoing uses feedback strategies to ensure that the desired task values, used by the whole-body controller, force the tasks to converge on the reference values provided by their trajectories. These servoing feedback controllers generally operate at frequencies between 100Hz and 10Hz. This control hierarchy of task planning, task servoing, and whole-body control is presented in Fig. 1.1. At each of these levels, a different abstraction of the control problem is employed. This is done to make the problem of converting high-level goals into joint torques more tractable, but entails the downside of producing tasks which may be incompatible and infeasible. Looking more deeply at each of these levels reveals why.

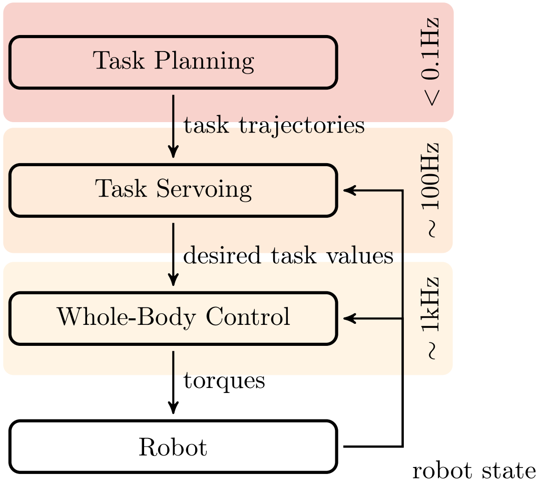


Figure 1.1 – A modern control hierarchy for highly redundant robotic systems, e.g. humanoid robots. At the lowest level is whole-body control, which determines the torques needed to accomplish a set of tasks. These tasks are controlled by the task servoing level where task trajectory errors are compensated using feedback. Finally the task trajectories are provided by high-level task planning, which is usually a combination of operator expertise and automated planning.

1.1.1 Task Planning

Planning, or more specifically, *motion planning*, in humanoid robotics focuses primarily on the problem of finding a collision free path in *configuration space* from one configuration to another: getting from point A to B. To do so, a map of the robot's environment is projected into the configuration space, and search techniques such as A*, or *Rapidly exploring Random Tree*, (RRT) are used to find a collision free path [Lavalle, 1998]. This path is then smoothed using interpolation and provided as a joint-space task trajectory to a low-level whole-body controller.

Kinodynamic Planning Unfortunately, not all configurations are stable, and static equilibrium criteria need to be added to the search to make sure the joint-space task does not cause the robot to lose balance [Kuffner et al., 2002]. Using preplanned footsteps, Yoshida et al. [2005a] project the operational-space feet tasks into the configuration space, which enables walking motions to be planned. However, these planners only consider kinematics and accounting for dynamics is impossible. As such, a decoupled approach, where a path is planned and used as an input to a reactive controller, such as a *Zero Moment Point* (ZMP) generator, is adopted by [Yoshida et al., 2005b; Harada et al., 2008]. This so-called *kinodynamic planning* can produce reasonably stable joint-space tasks but the dynamic model used by the ZMP generator oversimplifies the true dynamics of bipedal locomotion. Therefore, the resulting whole-body motions are typically not stable when executed in an open-loop.

Planning with Whole-Body Dynamics Planning while accounting for the whole-body dynamics is very time consuming and typically intractable without a good initial guess for the plan [Bouyarmane and Kheddar, 2012; Ibanez et al., 2017]. To counter this, Dai et al. [2014] propose a planning scheme which uses the robot's centroidal dynamics, rather than the whole-body dynamics, to attain highly dynamic motions in minutes to hours of computation. These motions however do not consider the actuator limits and are likely impossible on a real platform. *Time-Optimal Path Parameterization* (TOPP) uses the whole-body dynamics and can efficiently compute bang-bang style time laws for predefined paths, but cannot account for multiple tasks [Pham, 2014; Pham et al., 2016].

Incorporating Additional Tasks To extend planners to incorporate or consider other tasks than just those involved in walking, [Weghe et al., 2007], [Berenson et al., 2009b], [Dalibard et al., 2009], [Berenson et al., 2009a], [Berenson et al., 2011], and [Cognetti et al., 2014] all propose methods of projecting operational-space tasks or constraints into the configuration space. These works, however, do not take into account the many gains and parameters of the whole-body controller, which greatly affect the final whole-body motion.

Operator Intervention Most of the aforementioned studies focus solely on moving the robot from one point to another without hitting any obstacles, but many tasks are planned manually. Object manipulation and grasping are often planned entirely by expert operators. The best example of this is the DARPA Robotics Challenge, where robots were controlled using both automated planning techniques and human intervention. A review of the various intervention techniques is provided by [Yanco et al., 2015]. Planning done by humans is of course prone to human error, and the vague models used to plan task trajectories are based on human motor control intuition, which is generally unsuited to robot control. Regardless of the planning technique, the underlying whole-body controller and task servoing layers are ignored in order to make the problem manageable. Unfortunately, ignoring these levels leads to incompatible and infeasible tasks.

1.1.2 Task Servoing

Assuming task trajectories have been planned using some automated method or by an expert operator the next step in the control hierarchy is to apply some task specific control law or policy to the servoing of the task trajectories. This is done because any open-loop control scheme is bound to build up error and drift, so simple feed-forward control, i.e. passing the planned task trajectories directly to the whole-body controller, is not enough to ensure the tasks correct any accumulated errors.

PID Controllers The most basic and common task servoing controllers are *Proportional-Integral-Derivative* (PID) controllers and their variants: PI, PD. These are easily implemented for any task and will ensure convergence to a set reference given appropriate selection of the gains, and that the reference can be attained. Adding a feed-forward term to PID controllers has been shown to improve convergence [An et al., 1986], assuming the task trajectory dynamics are feasible. From a practical standpoint, such task servoing controllers can also serve to convert position and velocity signals into acceleration signals which are required when using inverse dynamics control. Unfortunately, simple PID controllers only ensure the asymptotic convergence of the task that they are servoing, and this may not even be possible given the other tasks and constraints. For tasks such as those involved in walking more complex feedback controllers are needed to stabilize the task dynamics.

MPC and LQR *Model Predictive Control* (MPC) can improve the robustness of task execution in the face of incompatibilities and infeasibilities by repeatedly recomputing an optimal control trajectory over a given horizon of time using *Linear Time-Invariant* (LTI) models of the desired task dynamics. One of the most prolific usages of task-level MPC is the generation of stable walking patterns for bipedal locomotion [Ibanez et al., 2017]. Given a ZMP trajectory and footstep locations, the MPC algorithm produces a *Center of Mass* (CoM) trajectory, which should ensure stable walking [Kajita et al., 2003, 2006; Wieber, 2006]. These algorithms are typically real-time or close to real-time, and certain assumptions can yield closed form solutions which are even faster [Tedrake et al., 2015]. As an alternative to (computationally costly) MPC algorithms, Linear Quadratic Regulators (LQRs) can also be used to control CoM tasks for locomotion [Kuindersma et al., 2014]. The LTI dynamic models used in both LQR and MPC work well for walking, but are not sufficiently expressive to account for tasks other than a CoM tracking. Extensions of these controllers can be used to determine optimal foot-placement (i.e. operational-space task trajectories for the feet) in addition to the CoM trajectory [Ibanez et al., 2014; Deits and Tedrake, 2014]. But stringent convexity requirements keep such methods from being extended to generic multi-task optimization.

The Limits of Task Servoing Regardless of the task servoing controller used, no current method allows for real-time global optimization of the task trajectories based on the overall execution of the whole-body motion. Furthermore, all task servoing controllers rely on the task references provided by the high-level task planning. If those trajectories are impossible to execute on the real system, then in the best case scenario, the servoing controllers will simply prevent the robot from executing the task because it will render the motion unstable. This is problematic because if the task was specified, then it should probably be executed. In the worst case scenario, the servoing controller will aggressively try to correct tracking errors, and this could lead to dangerous whole-body motions for both the robot and any bystanders.

1.1.3 Whole-Body Control

With the planned task trajectories, and their servoing controllers, the final and lowest-level in the control hierarchy is whole-body control. Whole-body controllers reactively calculate the joint torques, necessary to accomplish some set of tasks expressed in either operational-space or joint-space, for all of the actuated *Degrees of Freedom* (DoF) of the given robot, while respecting physical constraints. We restrict our overview of whole-body control to *Inverse Dynamics* (ID) whole-body control because it allows compliant interaction with the environment to be achieved. *Inverse Velocity Kinematics* (IVK) whole-body control is also possible, but not the focus here.

Whole-Body Control Challenges The concept of whole-body control is first concretized by [Khatib et al., 2004; Gienger et al., 2005; Sentis and Khatib, 2005], using a multi-task based approach to producing complex whole-body behaviors. The main advantages of humanoids such as, multi-tasking, compliant multi-contact interaction, and floating-base dynamics, are also the primary challenges in whole-body control. Dealing with the complexities of floating-base dynamics and compliant contacts is not trivial many works in whole-body control tackle these issues [Sentis and Khatib, 2005; Aghili, 2005; Mistry et al., 2010; Righetti et al., 2011].

The Need for Prioritization Being able to execute multiple simultaneous tasks using all of the DoF is the key to complex whole-body motions, but combining tasks, as we have already stated, leads to conflicting control objectives if they are incompatible and/or infeasible. This is such a challenge in fact, that in all the aforementioned works, the tasks are combined using hierarchical null-space projection techniques. Hierarchies ensure that important tasks such as the CoM tasks, which manage balance, are unperturbed by unimportant tasks, like postural tasks. However, sometimes, multiple tasks are important and not doing one of them is unacceptable. For example we take the classic example of a balancing CoM task, and a hand reaching task, where the robot has to hit a button to stop a nuclear apocalypse. Prioritization is therefore necessary, but picking priorities is not always a simple issue.

Dealing with Constraints Beyond multi-tasking, whole-body control has to respect the system constraints. Determining an appropriate distribution of contact forces for a given CoM state requires delicate treatment of the equilibrium constraints and whole-body dynamics [Pratt and Pratt, 1998; Hyon et al., 2007; Stephens and Atkeson, 2010; Righetti et al., 2013]. With nullspace projection methods, equality constraints can be handled via Lagrange multipliers, but inequality constraints, like friction contacts and actuator limits, must be converted into objective functions [Dietrich et al., 2015].

Numerical Optimization Techniques Techniques from numerical optimization offer another route for solving robot control problems, because inequality constraints can be directly included into the problem formulation [Kanoun et al., 2009; Decre et al., 2009]. Tasks become objective functions and are formulated as quadratic errors between the joint-space projection of the task and the desired task value. If the constraints are affine, then the problem is formulated as a *Quadratic Program* (QP) [Salini et al., 2011; Saab, 2011; Bouyarmane and Kheddar, 2011]. More details on QPs are provided in Chapter 3. Formulating the whole-body controller as a QP allows rigid contact constraints to be accounted for directly as shown by Salini [2012]; Saab et al. [2013], and bipedal equilibrium can be robustly integrated into the whole-body controller [Herzog et al., 2014]. An interesting corollary to QP based control is the LQR whole-body controller developed by Mason et al. [2014], which can be looked at as a hybrid of the task servoing and whole-body control layers. Formulating the control problem directly as an optimization problem affords some flexibility in the mixing of tasks, or objective functions. As with projection methods, tasks may be accomplished hierarchically using a cascade of QPs as presented by [Escande et al., 2014]. Alternatively, tasks can be combined using "soft" prioritization techniques, where the importance of the task is a continuous value from 0 to 1. Such methods have been explored by [Salini et al., 2011; Saab, 2011; Bouyarmane and Kheddar, 2011; Liu et al., 2015b].

Modeling Issues Despite all of these advancements, whole-body control is a reactive model-based control method and therefore prone to modeling errors and tracking drift. Work by Feng et al. [2014] attempts to reject modeling errors on a real humanoid system by mixing IVK and ID whole-body controllers. Prete and Mansard [2016] investigate the robustness of whole-body control regarding torque tracking errors (an important real-world consideration) and show that model uncertainties can generate unstable behaviors. They also go on to propose a method of including these uncertainties directly in the whole-body controller. In addition to model uncertainties and errors, whole-body control is reactive in nature, and solves a new control problem at each time step. It does not use the horizon of future control objectives to predict the future states of the robot, and because of this, can get in states where the constraint set becomes infeasible.

1.2 The Problem with Modern Control Architectures

At each level in the control hierarchy, assumptions are made about the control problem to make the problem scope reasonable. At the whole-body controller level, the model is highly complex but generally does not include complex non-linear effects like articular friction and model uncertainties. To maintain the convexity of the whole-body control problem, the whole-body dynamics must be recomputed at each control instant. This sort of reactive whole-body control can only optimally minimize task errors at an instant in time and has no knowledge of future task objectives and constraints. This makes reactive whole-body control myopic with regard to the whole-body motion. One consequence of this myopia is getting "stuck" in constrained states and configurations as in Fig. 1.2a.

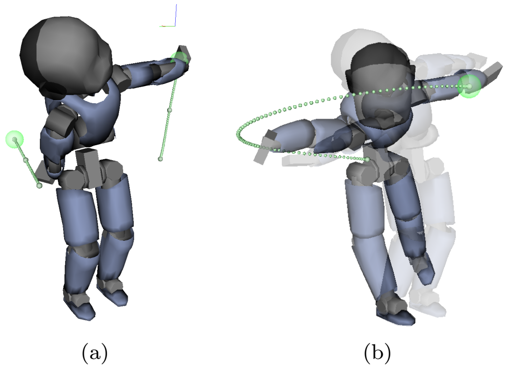

(a) (b)

Figure 1.2 – Incompatible and infeasible tasks which result in (a) the robot getting stuck in a constrained configuration and (b) a loss of equilibrium.

At the task servoing level, the whole-body dynamics and kinematics are mostly ignored in order to render the task servoing problems tractable in near real-time. Task servoing controllers use LTI models as simplifications of the task dynamics. These simple models enable rapid computation of predictive and reactive control. However, it is generally because the tasks are planned individually with simplified models that they cannot be executed properly. Thus, task servoing can render the low-level control more robust to divergence, but if the task trajectories being servoed are impossible, then the task servoing can lead to unwanted motions.

Moving up the hierarchy, automated task planning uses kinematic and possibly dynamic models of the robot and its environment, to convert high-level objectives into task trajectories. Alternatively, the task planning could be done by an expert operator, and therefore, it is the operator's model of the robot which is used to devise the task trajectories. Regardless of the method, task planning is too slow or too simplistic to account for the two lower levels of control, complex whole-body dynamics, and high-level objectives, simultaneously. For this reason, it cannot be used on-line to replan tasks when they are not executed as expected.

The result of these various assumptions is a set of task trajectories, which may or may not be *compatible* with one another, and/or *feasible* with the system constraints. Incompatible and infeasible tasks result in unexpected and generally undesirable whole-body motions such as a loss of balance due to reaching as in Fig. 1.2b.

The Missing Link Ultimately, what is missing is a way to automatically modify task trajectories using feedback from the execution of the whole-body motion, in order to maximize their compatibility and feasibility without fully re-planning them. This missing component of the control hierarchy is presented in Fig. 1.3. Thus, **the primary objective of this work develop this layer of task compatibility and feasibility maximization** but in order to do this, we first need to understand how others have attempted to solve similar issues.

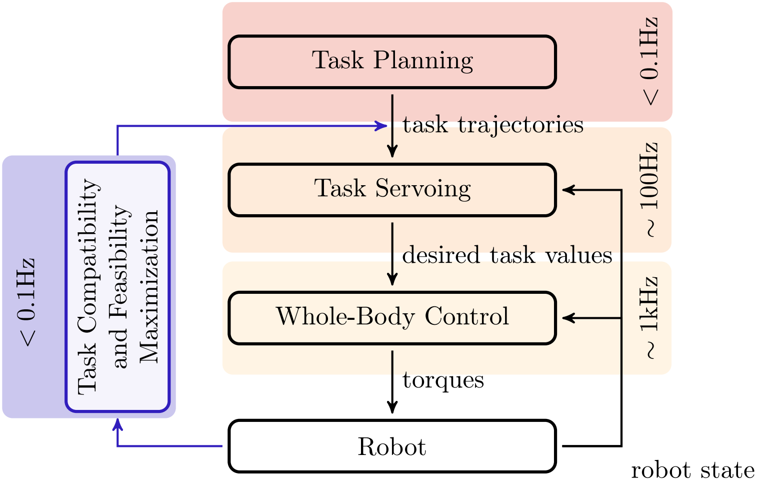


Figure 1.3 – The task compatibility and feasibility maximization loop proposed in this paper is designed to correct incompatible and infeasible tasks produced by this modern control architectures.

1.3 Related Work

The goal of understanding and improving task compatibility and feasibility is a relatively new one, which has arisen from improvements in humanoid whole-body control. Bouyarmane and Kheddar [2015] are the first to consider the numerical problems which arise from treating whole-body control as a multi-objective constrained optimization problem. In their work, they show how under certain weighting conditions, asymptotic stability can be ensured.

The concept of *viability* for humanoids, which is first introduced in [Wieber, 2000] and elaborated in [Wieber et al., 2016], defines the set of states from which a feasible control input can be found to keep the robot from entering states of disequilibrium. Viability is a restriction of *controllability* which simply says that it is possible to find a feasible control input to move from some starting state to some final state. With regards to task compatibility and feasibility, the concept of viability can be seen as a whole-body corollary to the problem of task feasibility. In task feasibility, the goal is to determine the set of feasible task trajectories (inputs), which move the tasks from their initial states, to some specified goal states. Unfortunately, viability is typically intractable to compute for systems like humanoids.

In [Wieber et al., 2017], the nature of why task objectives cannot be realized simultaneously on a redundant system is posed as a problem of linear dependency. To solve this for a 2 DoF system, “trust region” bounds are placed on the objective variable, which keep the objectives from becoming linearly dependent. To the best of our knowledge no other works have directly addressed the issue of task compatibility and feasibility.

Although few works consider the problem of incompatible or infeasible tasks as it is described here, these issues manifest themselves as performance degradation in terms of task trajectory tracking. The more compatible and feasible the tasks are, the better the robot will follow the task trajectory references, up to its precision limits. Thus, any work related to improving the control performance of robots using task-based control, has directly or indirectly attacked this problem, and can be lumped into two basic categories:

1. Works which modify the controller and/or its parameters to *reject* the effects of task incompatibility and/or infeasiblity.
2. Works which modify the task trajectories to *remove* task incompatibility and/or infeasiblity.

In the case where task trajectories cannot be modified, the incompatibilities and infeasibilities associated with them will exist and produce undesirable behaviors. These effects can be attenuated, and in some cases removed completely, through modification of the underlying controller and its parameters, e.g. weight, hierarchies, and gains. When the task trajectories can be modified, it may be possible to render tasks both compatible and feasible, thus mitigating the need to tune controller parameters. Each of these methods have their pros and cons, but the endgame is the same, make the robot execute the desired behavior properly, i.e. track the task trajectory references accurately.

1.3.1 Manipulator Performance Measures

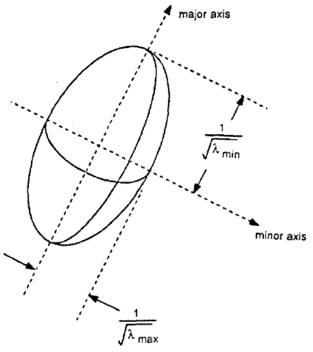

major axis
$\frac{1}{\sqrt{\lambda_{min}}}$
minor axis
$\frac{1}{\sqrt{\lambda_{max}}}$

Figure 1.4 – Velocity ellipsoid described by Chiu [1987].

In the earliest robotics literature, the main causes of tasks not being executed properly are kinematic singularities.

Avoiding Singularities In this case, tasks are infeasible with respect to geometrical constraints, and the objective is to find feasible task trajectories. The performance measure developed in [Salisbury and Craig, 1982] is based on the workspace volume and singularities associated with manipulation tasks. They show that maximizing the number of *isotropic points* in the workspace can minimize error propagation in tasks. Isotropic points are configurations where the columns of the Jacobian are orthogonal and equal in magnitude. Klein and Huang [1983] consider the numerical divergence associated with pseudoinverse control of redundant manipulators, first proposed by Whitney [1969], and build on the work of [Liegeois, 1977] to approximate a minimax solution for the inverse velocity kinematics. Baillieul [1985] looks at an alternative approach to the non-closedness of pseudoinverse control by proposing an extended Jacobian technique for redundancy resolution, which explicitly avoids singularities both kinematically and algorithmically.

Improving Performance Measures Locally Yoshikawa [1984] presents the first *manipulability measure*, which measures the ability of a manipulator to move in any direction in operational-space. This work is elaborated in [Yoshikawa, 1985a], and then adapted to task dynamics in [Yoshikawa, 1985b]. The *dynamic manipulability*, is then the relationship between joint torques and the end-effector (EE) acceleration. Chiu [1987] first attributes the word *task compatibility* to mean the appropriateness of a posture for performing either fine or coarse movements. Here a manipulator is viewed as a mechanical transformer and compatibility is a function of the transmission ratios defined by the *velocity ellipsoid* (see Fig. 1.4), and *force ellipsoid*. A compatibility index is computed from these ratios, which is maximized in the control by adding a low priority joint-space task, yielding human-like postures for tasks like writing. A review of these measures and their application to redundant manipulators is provided by [Klein and Blaho, 1987]. Throughout these works, gradient minimization tasks are used to maximize the manipulability, and dynamic manipulability ellipsoid volumes in a reactive manner, using a secondary task. This can help keep reactive controllers from getting too close to singularities and joint limits, but without a view of the whole task trajectory, it cannot ensure global task feasibility.

Improving Performance Measures Globally Lee and Lee [1988], encode task trajectories as manipulability ellipsoids, which are then maximized in volume, thus optimizing the singularity avoidance of the task trajectory. Cloutier et al. [1994] introduce the task conformance index which takes into account the task constraints as well as the robot constraints. These measures are then used to globally optimize task feasibility in [Kapoor et al., 1998]. In [Lee, 1989], it is shown that application of the manipulability measures to a dual manipulator scenario system is non-trivial due to added kinematic constraints of the arm interactions. Furthermore, application of the many performance criteria is ill-defined for the humanoid case, where there are many tasks being executed simultaneously and on a free-floating base. Consequently, alternative forms of trajectory optimization must be explored to ensure tasks are feasible.

1.3.2 Trajectory Optimization & Optimal Control

Trajectory optimization is a large field which is tightly coupled with the domain of optimal control, [Kirk, 2012]. Focusing on robotics, most of the earliest works in trajectory optimization revolve around finding joint-space or operational-space task trajectories which maximize or minimize some optimality criterion, while respecting the robot constraints.

Manipulator Trajectory Optimization From a task compatibility and feasibility perspective, this corresponds to tasks which are globally feasible given the constraints, and optimal according to tracking criteria. For example, Bobrow et al. [1985] explore time optimal trajectories for a specified path under dynamic constraints, e.g. minimum and maximum torques actuator torques. In [Nakamura and Hanafusa, 1987], Pontryagin's maximum principle is used for globally optimal redundancy resolution using kinematic and dynamic performance indexes as optimality criteria. Mayorga et al. [1995] find the optimal trajectory for obstacle avoidance via potential fields (see [Khatib, 1986]) and singularity avoidance. In [Hayward et al., 1994], the energy optimal path and trajectory is found for a given set of operational-space waypoints and the robot's constraints. In [Field and Stepanenko, 1996] energy-optimal trajectories are determined using a variation of dynamic programming which scales linearly with the number of DoF. Regrettably, the majority of these techniques result in trajectories which saturate the actuators (a consequence of bang-bang control), and may result in poor tracking at run-time. Chettibi et al. [2004] therefore propose the use of additional acceleration and jerk constraints in joint-space and simplify the optimal control problem using clamped cubic splines.

Humanoid Trajectory Optimization These methods, unfortunately, do not scale well to high-dimensional systems because they require that the entire control horizon be integrated. Given changing contacts, discontinuous dynamics, and the high-dimensionality of humanoid robots, most classical optimal control approaches cannot be applied to humanoids. Approximating contact dynamics as smooth processes can alleviate some of these difficulties, but results in potentially unrealistic overall behaviors [Todorov, 2011]. Kolter and Ng [2009] render optimal control efficient by reducing the floating-base representation to fixed base kinematic chains and considering only spline-parameterized operational-space trajectories. Local approximations of the dynamics can also be used to circumvent the dimensional explosion of global methods, such as the iterative *Linear Quadratic Gaussian* (LQG) control developed by Todorov and Li [2005], and then extended to stochastic optimal control by Toussaint [2009]. Using simple optimality criteria and a spline parameterization of a joint-space trajectory, Borno et al. [2013] produce highly dynamic movements on a simulated humanoid with changing contacts, but the optimizations take between 1 and 15 hours. Posa et al. [2014] and Neunert et al. [2017] show that formulating the problem as a *Sequential Linear Quadratic* (SLQ) program can achieve global trajectory optimality (see Fig. 1.5) but limits the types of constraints which can be used in the problem. Recently, *direct collocation* methods have been used to alleviate these constraint restrictions, but still require minutes of computing time even for short horizons [Posa et al., 2016; Pardo et al., 2017].

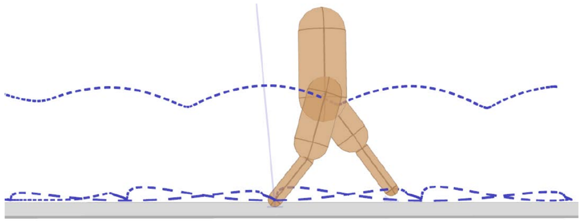


Figure 1.5 – Example of optimal task trajectories from [Neunert et al., 2017].

Modeling and Dimensionality Issues Model-based methods such as these are subject to modeling errors and uncertainties, which can render them unstable. *Adaptive control* methods simultaneously estimate model parameters while performing control, allowing the controller to compensate for modeling errors [Åström, 1983; Craig et al., 1987]. However, parameter convergence is not always guaranteed [Boyd and Sastry, 1983]. *Robust control* allows model uncertainty bounds to be taken into account in the control problem, but the output is sensitive to the chosen bounding [Slotine, 1985]. None of these methods, however, account for the underlying task servoing and whole-body control layers, nor any additional tasks which may be active. In order to maximize task compatibility and feasibility, errors in the model must be compensated and task trajectories need to be re-planned in a closed-loop manner, while accounting for the underlying control hierarchy and the existence of multiple tasks. Regrettably, the dimensionality of systems like humanoids creates prohibitively large state-spaces and limits the use of global optimization methods to open-loop high-level task trajectory planning. By only looking at a short control horizon rather than the entire control time line, whole-body non-linear MPC can reduce the dimensional complexity of the optimal control problem.

1.3.3 Whole-Body Non-Linear Model Predictive Control

Given a set of tasks and constraints, whole-body *Non-linear MPC* (NMPC) attempts to find the optimal control signals for the current time step, by predicting the evolution of the robot over a short control horizon, using the whole-body non-linear time-varying dynamics. The output of whole-body NMPC is a trajectory of joint torques which optimally accomplishes the tasks over a horizon of time. The advantage of this approach is that the full model information can be exploited to optimally carry out all tasks thus optimizing performance criteria globally in a control sense. The disadvantage is the computational complexity. At a single time step the equations of motion can be considered linear; however, over a horizon of time, the dynamics are non-linear and the problem of predictive control becomes time-varying, non-linear, and non-convex. NMPC problems are computationally expensive to resolve and often hindered by local optima, but advances in non-linear optimization routines improve greatly on these points [Tassa et al., 2008; Allgöwer and Zheng, 2012; Grüne and Pannek, 2011].

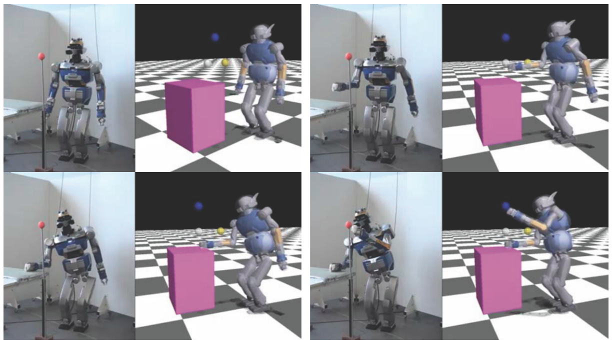


Figure 1.6 – Example of optimized multi-contact whole-body motion on the HRP-2 humanoid robot using whole-body NMPC from [Koenemann et al., 2015].

Implementation Concerns Despite the power of whole-body NMPC, in order to make it sufficiently computationally rapid for high DoF systems, many approximations must be made. Most notably, whole-body NMPC requires that the system dynamics be differentiated at various points along the whole-body trajectory. This is computationally costly and numerically unstable if the contacts are too realistic. Thus, the contact dynamics must be efficiently differentiated in order to use gradient-based trajectory optimization schemes [Todorov, 2011]. Beyond the major computational challenges, which are many, the control problem formulation is not evident. [Tassa et al., 2012] apply NMPC to a humanoid robot control problem and develop a custom cost function based on control and state costs. For their humanoid example, the state cost is essentially the combination of kinematic CoM and torso operational-space tasks. Using these costs (tasks), the robot starts from the ground and determines the torques needed to move to a standing position in simulation. These custom costs are formalized as tasks which can be specified by an operator in [Erez et al., 2013]. In [Tassa et al., 2014] a method of imposing control constraints (i.e. torque limits) on the NMPC problem is presented to keep humanoid robots from optimally tracking infeasible control trajectories. Koenemann et al. [2015] demonstrate an almost real-time implementation of these controllers on the real HRP-2 humanoid robot as shown in Fig. 1.6.

The Problem of Poorly Planned Tasks Whole-body NMPC is rapidly becoming a practical replacement for whole-body control and task servoing, but despite the gains in robustness from the closed-loop predictive scheme, whole-body NMPC is still sensitive to modeling errors. Furthermore, NMPC can optimally track all tasks, but if the task trajectories, or set points, are incompatible or infeasible, then NMPC is faced with the same dilemma as task servoing: either ignore the tasks because they are incompatible or infeasible, or try to track them aggressively and produce dangerous whole-body motions. So for all these advances, model-based optimization methods, for the moment, cannot solve two basic issues which produce incompatible and infeasible tasks. Firstly, they cannot modify the task trajectories themselves, only track them more or less. Secondly, they are subject to modeling inaccuracies which are part of why planned tasks are incompatible and infeasible in the first place. One alternative then is to forget the model all together and turn to model-free learning.

1.3.4 Model-Free Policy Search

In robotics, *Model-Free Policy Search* (MFPS) offers an interesting alternative to analytical model-based approaches like optimal control [Deisenroth et al., 2013; Kober et al., 2013]. The basic concept behind MFPS is that parameterized control strategies, or *policies*, can be improved incrementally to minimize (maximize) some cost (reward) attributed to the execution of the policy, or *rollout*. These rollouts can occur in simulation or in the real world. The benefit of this approach is that all of the modeling errors, un-modeled phenomena, and repetitive perturbations, which only appear when the whole-body motion is executed, can be captured by the cost attributed to the rollout. For the sake of simplicity, we use only the term “cost” for the remainder of this review, but it should be noted that costs and rewards can easily be interchanged. Using this cost, a policy update strategy, which is a numerical optimization routine in most cases, is used to optimize the policy parameters according to the rollout data. This process of *episodic learning* is iterated until some convergence criterion is met, e.g. a difference threshold between the previous and proposed policy parameter vector from the update.

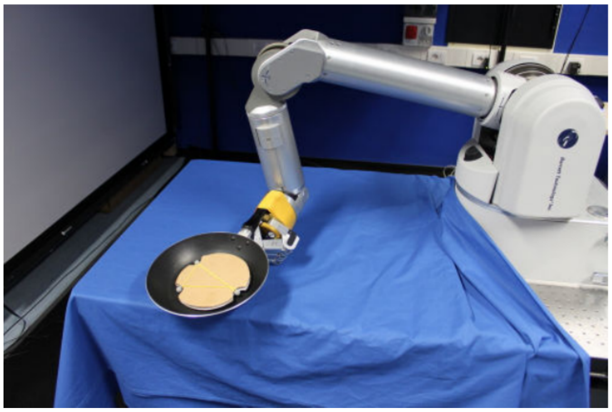


Figure 1.7 – Example of MFPS used to learn how to flip a pancake in [Kormushev et al., 2010].

Learning in Joint-Space Typically *Dynamic Movement Primitives* (DMPs) are used to parameterize the policies in joint-space [Ijspeert et al., 2013], and they are generally learned from demonstration [Billard et al., 2008; Argall et al., 2009][1]. With an initial DMP, MFPS is used to iteratively improve the policy (DMP) based on high-level cost functions. In [Peters and Schaal, 2008b] a manipulator learns to play tee-ball after 200-300 rollouts. Kober and Peters [2009] use MFPS to learn the ball-in-cup game over 75 rollouts. Kober et al. [2011] learn to play ping pong and throw a dart after 200-300 rollouts. Each of these works use specialized cost functions, which encode the high-level objective of the task they are trying to accomplish. For instance, Kober and Peters [2009] use the ball's distance from the cup as a cost function. Despite being able to achieve highly complex dynamic tasks, which might be impossible to program by hand, the MFPS in these studies requires many rollouts. This is because low-level joint-space control policies learned as part of the high-level task policy. In a preliminary work to [Kober and Peters, 2009], Kober et al. [2008] try to learn the joint-level PD controllers as part of the ball-in-cup policy. Here, it takes MFPS 500-600 rollouts in simulation to converge, as opposed to the 75 rollouts achieved on the real robot in their later work, where the joint PD controllers are not integrated as part of the learning. While this is an improvement, humanoids have many more DoF than manipulators, and learning policies in joint-space, even with low-level torque PD controllers, takes 500-1500 rollouts [Stulp et al., 2010].

Learning in Operational-Space Learning in operational-space rather than joint-space has been shown to improve MFPS efficiency [Peters and Schaal, 2008a; Kormushev et al., 2010], but trying to learn control policies for tasks as well as the low-level joint-space control is impractical for humanoids. Combining MFPS and low-level model-based control has been shown to reduce some of the learning complexity, and produce more efficient MFPS [Mühlig et al., 2009; Calinon et al., 2014; Calinon, 2016]. For example, Kalakrishnan et al. [2011] are able to learn complex manipulation skills (see Fig. 1.8) by learning control policies for wrench tasks which are then executed using IVK and ID controllers. Nevertheless, special considerations must be made for humanoids such as the fact that they can fall. Thus, MFPS needs to be as sample efficient as possible. To achieve this, different parameterizations can be used.

If a whole-body controller is exploited by the MFPS, then compact parameterizations can be

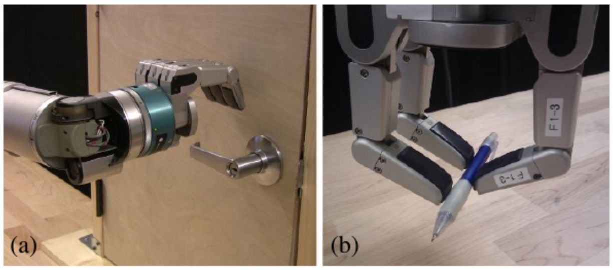


Figure 1.8 – Example of learned operational-space wrench tasks from [Kalakrishnan et al., 2011].

---

[1] See Sec. A.4 in the Appendix for more details on DMPs.

used to achieve complex task control policies. Modugno et al. [2016] use the task priorities in a whole-body controller as a policy parameterization, and learn the optimal task weights as a function of time for performing specific tasks. In [Spitz et al., 2017] a joint-space cost function within a whole-body controller is learned to avoid states where modeling errors make the control unstable. Mukovskiy et al. [2017] use DMPs to generate joint angle and CoM references for a walking pattern generator and a whole-body controller to re-plan walking trajectories online. While the results of these works show the benefit of combining MFPS with whole-body control, sample efficiency is still an issue.

Sample Efficient Learning If an underlying whole-body controller is used, then policies can be parameterized with fewer variables, and more efficient update strategies can be used. Most notably, *Bayesian Optimization* (BO) has become a favorite in humanoid robotics due to its sample efficiency [Mockus, 2012]. Calandra et al. [2014] use BO to determine the state-feedback gains for sagittal bipedal locomotion. In [Englert and Toussaint, 2016] the PR2 robot learns how to open a door using BO to optimize manipulation policy parameters. Marco et al. [2016] use BO to tune the weighting matrices of an LQR controller. Despite this sample efficiency, BO does not scale well to high dimensional policy representations and is limited therefore in terms of the policy complexity which it can learn. In [Antonova et al., 2016] and [Antonova et al., 2017], these limitations are pushed back by using specialized kernels which better capture the desired policy dynamics (bipedal locomotion in this case), and allow for clearer partitioning of the parameter space. Unfortunately, if the policy parameterization is made more complex, then more data, i.e. more rollouts, are needed to optimize the policy parameters.

1.3.5 Deep Learning Methods

Moving in the opposite direction, towards high-dimensional and highly expressive policy parameterization, *Deep Reinforcement Learning* (DRL) approaches attempt to learn *end-to-end* control policies (i.e. vision to actions) by training large *neural networks* over thousands of rollouts. Lillicrap et al. [2015] extend DRL to continuous control problems through the use of policy gradients, coining the term *Deep Deterministic Policy Gradient* (DDPG). [Lenz et al., 2015] propose the use of DRL to learn the dynamic model used in an MPC controller to cut food as shown in Fig. 1.9. In [Zhang et al., 2015], [Levine et al., 2016], and [Levine et al., 2017] manipulation skills are learned using visual feedback. In [Duan et al., 2016], a number of these algorithms are benchmarked for simulated control problems of various difficulty, but all require a vast amount of pre-training and rollouts to learn useful policies. [Gu et al., 2016] and [Gu et al., 2017] improve on the sample efficiency of these DRL methods but still require thousands of rollouts to learn policies, which may or may not be physically possible.

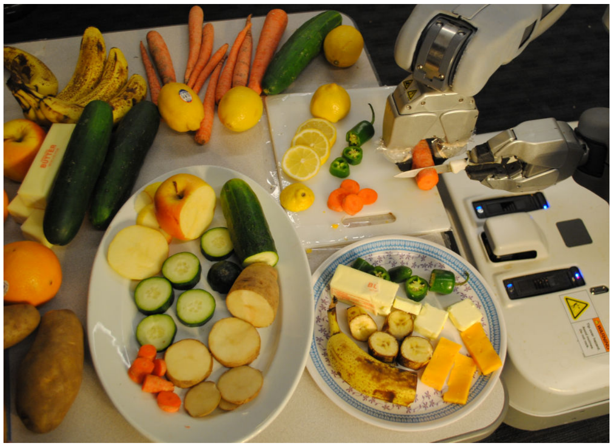


Figure 1.9 – The PR2 robot learning Deep MPC control for slicing food from [Lenz et al., 2015].

The Problem with DRL The benefit of DRL methods is that they have virtually limitless expressive potential in terms of the policy complexity they can learn. This potential comes at the cost of needing large quantities of data to properly train the deep networks and this is a serious practical problem. Even if the policies are learned in simulation, it is a certainty that they will not perform exactly as expected on a real robot. Accounting for these discrepancies may require hundreds or thousands of rollouts, which just is not possible on a humanoid [Koos et al., 2013]. These methods also have the issue of being very tailor made to specific activities: jumping, walking, swimming, etc. From a practicality standpoint, robots need to be able to do many things at once, and learning all these possible combinations is intractable. Some method is therefore needed to properly mix policies learned by DRL methods to accomplish useful tasks. For the purpose of maximizing task compatibility and feasibility, these methods could be used to entirely replace the control hierarchy presented in Fig. 1.1, and be adapted over time using cost functions to train the network to maximize task compatibility and feasibility. However, the number of rollouts needed to accomplish all of this would likely be prohibitive, and being in its infancy, DRL has yet to be demonstrated on a real humanoid.

1.4 Contributions

Modern whole-body control architectures have been proven to produce useful motions on complex real-world systems. The ability to specify tasks and constraints with these methods allows virtually any dynamically feasible whole-body motion to be achieved. In **Chapter 2**, a description of the whole-body controller used in this work is presented. The formulation is the one proposed by Salini [2012], and all of the relevant components of the task, constraint definitions as well as prioritization schemes are presented. More details on the controller and its software architecture are presented in [Eljaik et al., submitted].

Rather than ignore tried-and-true whole-body control, we attempt to build on its intuitive architecture and task/constraint specification methodology. Therefore, the objective of this work is to establish the task compatibility and feasibility maximization loop, shown on the left in Fig. 1.1. By closing the loop between low-level whole-body control and high-level task trajectory planning, modeling errors, assumptions, and uncertainties, which create incompatible and infeasible tasks, can be observed during the task execution and accounted for through optimization of the tasks. In this work, we show that by combining MFPS and model-based whole-body control architectures, we are able to close the task compatibility and feasibility maximization loop through trial-and-error learning, and produce useful whole-body motions on real humanoids in few trials. In order to reach this point, however, task compatibility and feasibility must be better understood.

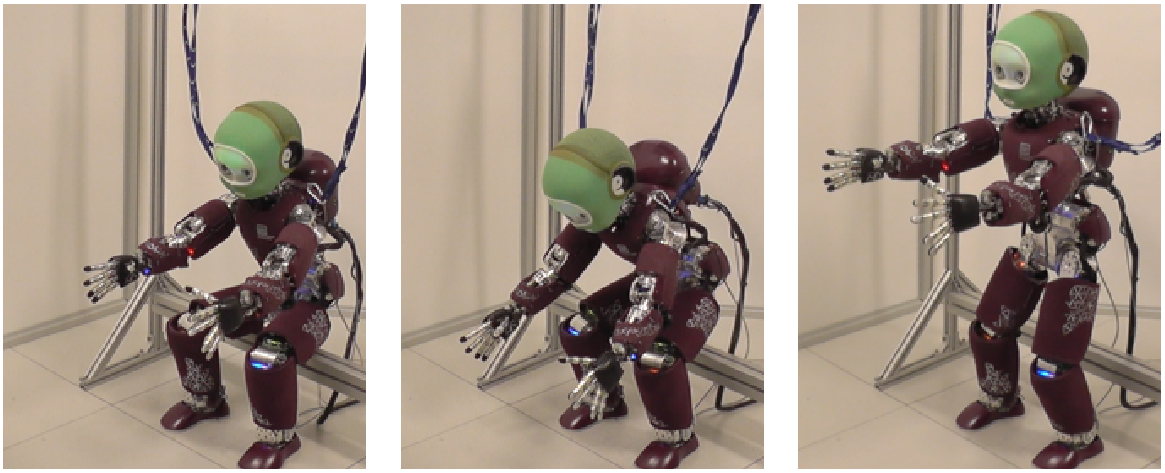
The first contribution of this work is to provide precise definitions of task compatibility and feasibility in the context of optimization-based whole-body control. Following the work of [Bouyarmane and Kheddar, 2015] and [Wieber et al., 2017], in **Chapter 3**, we look at the generic case of a multi-objective convex optimization problem and build a mathematical foundation for what it means for objectives to be compatible with one another and feasible with the problem constraints.

In **Chapter 4**, these tools are applied to a specific formulation of the ID whole-body control problem, and used to analytically measure task compatibility and feasibility. With these measures, a general model-based solution to rendering tasks optimally compatible and feasible is formulated and shown to be a complex NMPC problem. The complexity of the problem is discussed and leads to the realization that an efficient resolution may currently be untenable.

With an understanding of task compatibility and feasibility at a mathematical level, and the potential complexities of trying to resolve incompatibilities and infeasibilities using model-based techniques, the second major contribution is to redefine task compatibility and feasibility from a model-free standpoint. In **Chapter 5**, the task compatibility and feasibility problem is reformulated using a model-free perspective based on [Lober et al., submitted]. This formulation captures the underlying task servoing and whole-body controller in a generic parameterized control policy. The parameters of this policy are task relevant quantities which can be used to explore new control policies. A series of cost functions are introduced to measure the effects of incompatible and/or infeasible tasks, which are generic to the task and whole-body controller definitions. Using the model-free formulation and cost functions a model-free *Task Compatibility and Feasibility Maximization* (TCFM) algorithm is developed. The end result is a model-free method for optimizing task trajectories to maximize task compatibility and feasibility which is controller and task agnostic. The TCFM is designed to exploit the underlying model-based control architecture allowing for compact policy parameterizations. This leads to sample efficient learning designed to be implemented on real platforms to compensate for un-modeled phenomena.

In order to validate the proposed TCFM algorithm, we start by evaluating the different model-based task compatibility and feasibility metrics in **Chapter 6**. Using a series of scenarios with known task incompatibilities and infeasibilities, the metrics are tested to determine which are actually useful for quantifying and qualifying task compatibility and feasibility. With these metrics, the TCFM algorithm is validated in **Chapter 7**. Given the knowledge that the proposed method can truly maximize task compatibility and feasibility, it is tested on complex humanoid examples in **Chapter 8** [Lober et al., 2014]. The results from these tests indicate that the method is viable on a humanoid platform However, far too many trials are needed to optimize the tasks. To improve the sample efficiency, a new policy parameterization in combination with BO is used in **Chapter 9** [Lober et al., 2015, 2016]. Here, we show that by reducing the number of policy parameters and choosing an efficient update strategy, the TCFM is capable of solving complex task incompatibilities and infeasibilities in few trials. Finally, the parameterization is further refined in **Chapter 10** and demonstrated on a real humanoid [Lober et al., submitted]. The results from these experiments prove that the MFPS method is both controller agnostic and feasible on real platforms.

Because of the trial-and-error nature of MFPS, the proposed method is only as fast as the motion being optimized. Thus it does not operate at the desired frequency of 0.1Hz as shown in Fig. 1.3. Nevertheless, by combining analytical model-based controllers with data-driven MFPS, we are able to solve problems which would be otherwise intractable using simply one or the other — e.g. producing dynamically complex motions on real robots, like the example shown in Figs. 1.10a-1.10c. In **Chapter 11**, these and other conclusions on the body of work and future perspectives are presented to the reader.



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

Figure 1.10 – In (a), (b), and (c), a time-lapse is shown of an optimized standing motion performed by an iCub robot, which was found using the task compatibility and feasibility maximization algorithm developed in this work. The un-optimized motion caused the robot to fall.

1.4.1 List of Publications

[Lober et al., 2016]	R. Lober, V. Padois, and O. Sigaud. Efficient reinforcement learning for humanoid whole-body control. In *IEEE-RAS 16th International Conference on Humanoid Robots*, pages 684–689, Nov 2016
[Lober et al., 2015]	R. Lober, V. Padois, and O. Sigaud. Variance modulated task prioritization in Whole-Body Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3944–3949, Sept 2015. URL `http://dx.doi.org/10.1109/IROS.2015.7353932`
[Liu et al., 2015a]	M Liu, R Lober, and V Padois. Whole-Body Hierarchical Motion and Force Control for Humanoid Robots. *Autonomous Robots*, Special issue on Whole-body control for Humanoid Robots:1–13, 2015a
[Lober et al., 2014]	R. Lober, V. Padois, and O. Sigaud. Multiple task optimization using dynamical movement primitives for whole-body reactive control. In *IEEE-RAS International Conference on Humanoid Robots*, pages 193–198, Nov 2014

submitted

[Lober et al., submitted]	R. Lober, J. Eljaik, G. Nava, S. Dafarra, F. Romano, D. Pucci, S. Traversaro, F. Nori, O. Sigaud, and V. Padois. Optimizing task feasibility using model-free policy search and model-based whole-body control. In *IEEE International Conference on Robotics and Automation*, submitted
[Eljaik et al., submitted]	J. Eljaik, R. Lober, A. Hoarau, and V. Padois. OCRA - Optimization-based Controllers for Robotics Applications. *Frontiers in Robotics and AI: Humanoid Robotics*, Building the iCub Mindware: Open-source Software for Robot Intelligence and Autonomy, submitted
[Romano et al., submitted]	F. Romano, G. Nava, M. Azad, J. Camernik, S. Dafarra, O. Dermy, C. Latella, M. Lazzaroni, R. Lober, M. Lorenzini, D. Pucci, O. Sigaud, S. Traversaro, J. Babic, S. Ivaldi, M. Mistry, V. Padois, and F. Nori. The CoDyCo Project achievements and beyond: Towards Human Aware Whole-body Controllers for Physical Human Robot Interaction. *IEEE Robotics and Automation Letters*, Special Issue: Human Cooperative Wearable Robotic Systems, submitted

Chapter 2

Reactive Whole-Body Control

Whole-body control is at the core of this work. In this chapter, we present a detailed description of the whole-body controller used here in order to lay the groundwork for the rest of the study. First, an overview of the free-floating parameterization is presented, followed by task and constraint formulations, prioritization strategies, and finally the task servoing controllers. These details later allow us to formulate a mathematical notion of what it means for tasks to be compatible and feasible.

In this chapter, the primary reactive whole-body controller used in this work is presented. This controller follows the design laid out by Salini et al. [2011] and Bouyarmane and Kheddar [2011]. The reader is directed to these work for more details on the minutia of the controller. The primary concern of this summary is to present the necessary equations and relationships to understand the critical aspects of the controller used. Here, an optimization-based controller is developed, which formulates the control problem as one of minimizing control objective functions while respecting the system constraints. Specifically, the problem is formulated as a Quadratic Program (QP) using the second-order rigid body dynamics of the robot. Therefore, the control objectives are expressed as either accelerations, torques or wrenches, allowing for complex dynamic interactions with the environment. The control constraints are expressed directly in the QP as affine equalities and inequalities. For the purposes of the controller formulation, details on convex optimization are omitted in this chapter, as they are not required for understanding the overall controller structure. However, convex optimization is studied more closely in Chapter 3.

The most generic form of the whole-body controller studied here can be summarized by the following optimization problem,

$$\underset{\boldsymbol{\chi}}{\arg\min} \quad f^{\text{task}}(\boldsymbol{\chi}) \tag{2.1}$$

$$\text{s.t.} \quad G\boldsymbol{\chi} \leq \boldsymbol{h} \tag{2.2}$$

$$A\boldsymbol{\chi} = \boldsymbol{b}. \tag{2.3}$$

The objective, $f^{\text{task}}(\boldsymbol{\chi})$, is a function of the optimization variable, $\boldsymbol{\chi}$, and is determined by control objectives, or tasks. The resolution of the objective is subject to (s.t.) the inequality and equality constraints in (2.2) and (2.3) respectively, which ensure that the control constraints are respected. We begin this chapter with a brief description of the free-floating rigid body dynamics of the system. The parameterization of the dynamics forms the optimization variable. The control objectives, or tasks, and constraints are then detailed and written in terms of the optimization variable. Finally, task prioritization schemes are discussed.

2.1 Free-Floating Rigid Body Dynamics

For robots whose root link can float freely in Cartesian space, e.g. humanoids, it is necessary to consider the pose of the root link with respect to (w.r.t.) the inertial reference frame. The primary method for doing so is to account for the root link pose directly in the generalized coordinates, $\boldsymbol{q}$, of the robot as shown by [Sentis and Khatib, 2005; Mistry et al., 2010; Righetti et al., 2011]. The generalized configuration parameterization for floating base robots,

$$\boldsymbol{q} = \begin{Bmatrix} \boldsymbol{\xi}_b \\ \boldsymbol{q}_j \end{Bmatrix}, \tag{2.4}$$

therefore contains the pose of the base link w.r.t. the inertial reference frame, $\boldsymbol{\xi}_b$, and the joint space coordinates, $\boldsymbol{q}_j$. Set brackets are used in (2.4) because $\boldsymbol{\xi}_b$ is a homogeneous transformation matrix in $\mathbb{R}^{4\times 4}$ and $\boldsymbol{q}_j$ is a vector in $\mathbb{R}^n$, with n the number of DoF of the robot, thus $\boldsymbol{\xi}_b$ and $\boldsymbol{q}_j$ cannot be concatenated into a vector. However, the twist of the base, $\boldsymbol{v}_b$, with the joint velocities, $\dot{\boldsymbol{q}}_j$, can be concatenated in vector notation, along with the base and joint accelerations to obtain,

$$\boldsymbol{\nu} = \begin{bmatrix} \boldsymbol{v}_b \\ \dot{\boldsymbol{q}}_j \end{bmatrix}, \quad \text{and} \quad \dot{\boldsymbol{\nu}} = \begin{bmatrix} \dot{\boldsymbol{v}}_b \\ \ddot{\boldsymbol{q}}_j \end{bmatrix}. \tag{2.5}$$

These representations provide a complete description of the robot's state and its rate of change, and allow the equations of motion to be written as,

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \underbrace{C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{q})}_{\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})} = S^\top \boldsymbol{\tau} + {}^eJ^\top(\boldsymbol{q})^e\boldsymbol{\omega}. \tag{2.6}$$

In (2.6), $M(\boldsymbol{q})$ is the generalized mass matrix, $C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}$ and $\boldsymbol{g}(\boldsymbol{q})$ are the Coriolis-centrifugal and gravitational terms, S is a selection matrix indicating the actuated degrees of freedom, ${}^e\boldsymbol{\omega}$ is the concatenation of the external contact wrenches, and ${}^eJ$ their concatenated Jacobians. Grouping $C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}$ and $\boldsymbol{g}(\boldsymbol{q})$ together into $\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})$, the equations can by simplified to

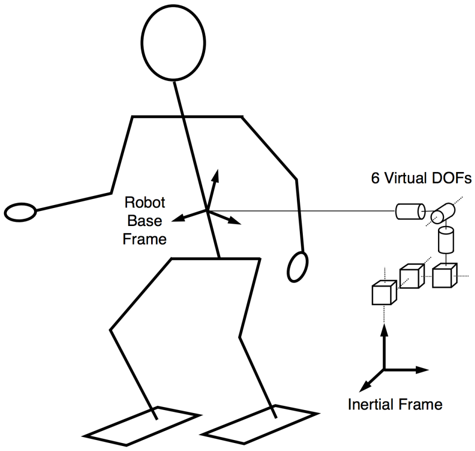


Figure 2.1 – A diagram indicating visually what it means to include the root link pose in the parameterization. The 6-DoF of the floating base are modeled as a 6-DoF linkage with the world or inertial frame. Image taken from [Mistry et al., 2010].

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu}) = S^\top \boldsymbol{\tau} + {}^eJ^\top(\boldsymbol{q})^e\boldsymbol{\omega}. \tag{2.7}$$

The joint torques induced by friction force could also be included in (2.7), but are left out for the sake of simplicity. Additionally, the variables $\dot{\boldsymbol{\nu}}$, τ, and ${}^e\boldsymbol{\omega}$, can be grouped into the same vector,

$$\boldsymbol{\chi} = \begin{bmatrix} \dot{\boldsymbol{\nu}} \\ \boldsymbol{\tau} \\ {}^e\boldsymbol{\omega} \end{bmatrix}, \tag{2.8}$$

forming the optimization variable from (2.1), and allowing (2.7) to be rewritten as,

$$\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^eJ^\top(\boldsymbol{q}) \end{bmatrix} \boldsymbol{\chi} = \boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu}). \tag{2.9}$$

Equation (2.9) provides an equality constraint which can be used to ensure that the minimization of the control objectives respects the system dynamics.

2.2 Control Objectives (Tasks)

The basic problem of control is to drive a system from some initial state to some desired state. The control of robots is no different, but the term state takes on greater ambiguity. For simple systems, such as the double integrator, linearized inverted pendulum, etc., state-space control is sufficient for virtually any high-level objective one could envision for the system. However, for a robot, describing the control problem solely in terms of its state, i.e. $\boldsymbol{q}$ and $\boldsymbol{\nu}$, is limiting and one may also want to describe it in terms of the pose and twist of an end-effector, or possibly even a wrench on some link (although not technically a state in the classical control sense). Far from being a detriment, this variability is what makes robots so useful but requires a bit of abstraction from classical state-space control vocabulary. For this reason, the term *task* is commonly used to indicate a control objective for a robot. Tasks, in second-order controllers, can be driven by desired accelerations, wrenches, or torques, and in operational-space or joint-space. They are expressed in the whole-body controller as functions of the errors between the desired and current values of the task. In this work, the square of the l^2-norm is used to create a quadratic objective function. Consequently, the task errors are expressed in the least-squares formulation.

2.2.1 Acceleration Task

Probably the most important, if not most prevalent, task is to move a link on the robot from one pose to another. Typically it is the end-effector(s) which are of interest, in the case of humanoids, the hands and/or feet. For the whole-body controller studied here, these tasks, which are generally expressed as desired positions or orientations, are converted to *acceleration tasks*, through means of task servoing. More details on task servoing are provided in Sec. 2.5. Once given a desired operational-space acceleration for a link, $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$, an acceleration task consists in finding the joint-space values which produce $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$,

$$\ddot{\boldsymbol{\xi}}_i^{\text{des}} = J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}, \tag{2.10}$$

where $J_i(\boldsymbol{q})$ and $\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})$ are the link Jacobian and its derivative. For the control objective, one simply rewrites the task as an error which must be minimized,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} - \ddot{\boldsymbol{\xi}}_i^{\text{des}} \right\|_2^2. \tag{2.11}$$

Using the squared l^2-norm produces a quadratic error term, which defines the objective function $f_i^{\ddot{\boldsymbol{\xi}}}$ to be minimized. The objective function $f_i^{\ddot{\boldsymbol{\xi}}}$ is then rewritten in terms of the optimization variable, $\boldsymbol{\chi}$,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| \begin{bmatrix} J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} - \left(\ddot{\boldsymbol{\xi}}_i^{\text{des}} - \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} \right) \right\|_2^2. \tag{2.12}$$

In (2.12) the term $\mathbf{0}$ represents a matrix of zeros. Regrouping terms as,

$$E^{\ddot{\boldsymbol{\xi}}} = \begin{bmatrix} J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix} \tag{2.13}$$

$$\mathbf{f}^{\ddot{\boldsymbol{\xi}}} = \ddot{\boldsymbol{\xi}}_i^{\text{des}} - \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}, \tag{2.14}$$

allows (2.12) to be written in the classical least-squares form as,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| E^{\ddot{\boldsymbol{\xi}}}\boldsymbol{\chi} - \mathbf{f}^{\ddot{\boldsymbol{\xi}}} \right\|_2^2. \tag{2.15}$$

The dependencies of $E^{\ddot{\boldsymbol{\xi}}}$ and $\mathbf{f}^{\ddot{\boldsymbol{\xi}}}$ have been removed for brevity. Acceleration tasks can be expressed in either joint-space or in operational-space. Here, the operational-space form is presented but the joint-space form can easily be produced as,

$$f_i^{\dot{\boldsymbol{\nu}}} = \left\| \dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_i^{\text{des}} \right\|_2^2, \tag{2.16}$$

with

$$E^{\dot{\boldsymbol{\nu}}} = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \tag{2.17}$$

$$\mathbf{f}^{\dot{\boldsymbol{\nu}}} = \dot{\boldsymbol{\nu}}_i^{\text{des}}, \tag{2.18}$$

where I is the identity matrix. Substituting (2.17) and (2.18) into (2.16) gives,

$$f_i^{\dot{\boldsymbol{\nu}}} = \left\| E^{\dot{\boldsymbol{\nu}}} \boldsymbol{\chi} - \mathbf{f}^{\dot{\boldsymbol{\nu}}} \right\|_2^2 . \tag{2.19}$$

2.2.2 Wrench Task

In order for robots to work properly in their environment, they must be able to interact with it. Not only does this allow the robot to manipulate and modify its environment, but it also allows the robot to exploit the environment to compensate for its underactuation and more generally to dynamically perform complex behaviors. Walking and balance are two pertinent examples of such behaviors because to achieve them, contact forces with the ground must be properly exploited. For details on this see [Park and Khatib, 2005], [Sentis et al., 2010], and [Righetti et al., 2013].

In order to interact with the environment, *wrench tasks* can be formulated to manage the interaction forces and torques,

$${}^{e}\boldsymbol{\omega}_i = {}^{e}\boldsymbol{\omega}_i^{\text{des}}. \tag{2.20}$$

where ${}^{e}\boldsymbol{\omega}_i^{\text{des}}$ is the desired external wrench to affect, and ${}^{e}\boldsymbol{\omega}_i$ is the wrench applied on the environment. Again, to formulate a control objective function, $f_i^{\boldsymbol{\omega}}$, the task is rewritten as the squared norm of a task error,

$$f_i^{\boldsymbol{\omega}} = \left\| {}^{e}\boldsymbol{\omega}_i - {}^{e}\boldsymbol{\omega}_i^{\text{des}} \right\|_2^2 . \tag{2.21}$$

Rewriting (2.21) in terms of $\boldsymbol{\chi}$ gives,

$$f_i^{\boldsymbol{\omega}} = \left\| \begin{bmatrix} \mathbf{0} & S_i^{\boldsymbol{\omega}} \end{bmatrix} \boldsymbol{\chi} - {}^{e}\boldsymbol{\omega}_i^{\text{des}} \right\|_2^2 , \tag{2.22}$$

where $S_i^{\boldsymbol{\omega}}$ is a wrench selection matrix which allows the i^{th} wrench to be controlled. Using,

$$\tag{2.23}$$

$$E^{\boldsymbol{\omega}} = \begin{bmatrix} \mathbf{0} & S_i^{\boldsymbol{\omega}} \end{bmatrix} \tag{2.24}$$

$$\mathbf{f}^{\boldsymbol{\omega}} = {}^{e}\boldsymbol{\omega}_i^{\text{des}}, \tag{2.25}$$

(2.22) can be written as,

$$f_i^{\boldsymbol{\omega}} = \left\| E^{\boldsymbol{\omega}} \boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\omega}} \right\|_2^2 . \tag{2.26}$$

2.2.3 Torque Task

Finally, it may also be desirable to specify *torque tasks* for purposes of regularization, among other possibilities. As with wrench tasks, torque tasks, can be written simply as,

$$\boldsymbol{\tau} = \boldsymbol{\tau}^{\text{des}}. \tag{2.27}$$

To formulate the control objective function, $f^{\boldsymbol{\tau}}$, the square norm of the task error is written,

$$f^{\boldsymbol{\tau}} = \left\| \boldsymbol{\tau} - \boldsymbol{\tau}^{\text{des}} \right\|_2^2 , \tag{2.28}$$

which can be reformulated in terms of $\boldsymbol{\chi}$ as,

$$f^{\boldsymbol{\tau}} = \left\| \begin{bmatrix} \mathbf{0} & S^{\top} & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} - \boldsymbol{\tau}^{\text{des}} \right\|_2^2 . \tag{2.29}$$

Once again regrouping terms,

$$E^{\boldsymbol{\tau}} = \begin{bmatrix} \mathbf{0} & S^\top & \mathbf{0} \end{bmatrix} \tag{2.30}$$

$$\mathbf{f}^{\boldsymbol{\tau}} = \boldsymbol{\tau}^{\text{des}}, \tag{2.31}$$

the least-squares form of the torque task is written,

$$f^{\boldsymbol{\tau}} = \| E^{\boldsymbol{\tau}} \boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\tau}} \|_2^2 \,. \tag{2.32}$$

2.3 Control Constraints

As with all real world control problems, there are limits to what the system being controlled can do. In this particular case, the main constraint is that of the system dynamics, i.e. the equations of motion. This means that any solution found must be dynamically feasible. Apart from this, the control input is typically bounded. For robots with revolute joints, this means that the torque which can be generated by the actuators is limited to plus or minus some value. Likewise, the joints themselves generally have limited operating ranges for various mechanical reasons. In addition to these common limiting factors, other phenomena such as unilateral and bilateral contacts can come into play.

2.3.1 Dynamics

The rigid body dynamics of the robot are governed by the equations of motion from (2.9). This constraint ultimately dictates the achievable dynamics of the system, and is formulated as the following equality constraint,

$$\underbrace{\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^{e}J^\top(\boldsymbol{q}) \end{bmatrix}}_{A^d} \boldsymbol{\chi} = \underbrace{\boldsymbol{n}(\boldsymbol{q}, \boldsymbol{\nu})}_{\boldsymbol{b}^d} \,. \tag{2.33}$$

The terms A^d and $\boldsymbol{b}^d$ are used to distinguish the equality constraint matrix and vector, respectively, for the dynamic constraints. This is done because the constraints presented here are combined by vertically concatenating them to form (2.3) and (2.2) in the controller.

2.3.2 Actuator Limits

Here, we assume that all articulations are revolute and therefore all actuation limits are torque limits, however, expression of force limits for prismatic joints would be another possibility. Writing these limits as an inequality provides an upper and lower bound on the amount of torque which can be exerted to accomplish the tasks.

$$\boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max}. \tag{2.34}$$

Expressing (2.34) in terms of $\boldsymbol{\chi}$ creates the following linear inequality,

$$\underbrace{\begin{bmatrix} \mathbf{0} & S^\top & \mathbf{0} \\ \mathbf{0} & -S^\top & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{\tau}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \boldsymbol{\tau}_{\max} \\ -\boldsymbol{\tau}_{\min} \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{\tau}}} . \tag{2.35}$$

2.3.3 Joint Limits

Probably the most common limitation of any robot is the range of motion which each joint can achieve. Whether linear or angular, most joints have a finite range through which they can move thus limiting $\boldsymbol{q}$. These joint limits can easily be expressed as a inequality on $\boldsymbol{q}$,

$$\boldsymbol{q}_{\min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{\max}. \tag{2.36}$$

Similarly to these position limits, we can also define limits on the joint velocities and accelerations,

$$\boldsymbol{\nu}_{\min} \leq \boldsymbol{\nu} \leq \boldsymbol{\nu}_{\max} \tag{2.37}$$

$$\dot{\boldsymbol{\nu}}_{\min} \leq \dot{\boldsymbol{\nu}} \leq \dot{\boldsymbol{\nu}}_{\max}. \tag{2.38}$$

The joint position limits, unlike the torque limits, must be manipulated somewhat in order to be properly expressed in $\boldsymbol{\chi}$. To formulate this constraint, $\boldsymbol{q}$ needs to be calculated while taking into account a second order prediction of the joint-space movement,

$$\boldsymbol{q}(t+h) = \boldsymbol{q}(t) + h\boldsymbol{\nu}(t) + \frac{h^2}{2}\dot{\boldsymbol{\nu}}(t), \tag{2.39}$$

where h is the prediction period, which is generally some multiple of the control period. Note that the floating base components of the configuration variable are not subject to articular limits, and their corresponding components in $\boldsymbol{q}$, $\boldsymbol{\nu}$, and $\dot{\boldsymbol{\nu}}$, are disregarded in (2.39). Dropping the time dependencies, the limits are written,

$$\begin{aligned} &\boldsymbol{q}_{\min} \leq \boldsymbol{q} + h\boldsymbol{\nu} + \frac{h^2}{2}\dot{\boldsymbol{\nu}} \leq \boldsymbol{q}_{\max} \\ \Leftrightarrow\ &\frac{2}{h^2}\left[\boldsymbol{q}_{\min} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \leq \dot{\boldsymbol{\nu}} \leq \frac{2}{h^2}\left[\boldsymbol{q}_{\max} - (\boldsymbol{q} + h\boldsymbol{\nu})\right]. \end{aligned} \tag{2.40}$$

Using $\boldsymbol{\chi}$, (2.40) can be rewritten as,

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{q}} \boldsymbol{\chi} \leq \underbrace{\frac{2}{h^2}\begin{bmatrix} \boldsymbol{q}_{\max} - (\boldsymbol{q} + h\boldsymbol{\nu}) \\ -\left[\boldsymbol{q}_{\min} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \end{bmatrix}}_{\boldsymbol{h}^{q}}. \tag{2.41}$$

From (2.41), one can of course naturally derive joint velocity and acceleration limits,

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\nu}} \boldsymbol{\chi} \leq \underbrace{\frac{1}{h}\begin{bmatrix} \boldsymbol{\nu}_{\max} - \boldsymbol{\nu} \\ -\left(\boldsymbol{\nu}_{\min} - \boldsymbol{\nu}\right) \end{bmatrix}}_{\boldsymbol{h}^{\nu}} \tag{2.42}$$

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\dot{\nu}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \dot{\boldsymbol{\nu}}_{\max} \\ -\dot{\boldsymbol{\nu}}_{\min} \end{bmatrix}}_{\boldsymbol{h}^{\dot{\nu}}}. \tag{2.43}$$

The choice of the prediction period, h, in the joint-space limits is crucial to the proper functioning of these constraints. Smaller values of h lead to more aggressive approaches to the joint limits, while larger values produce a more conservative treatment. This variability is due to the fact that the prediction does not take into account the deceleration capabilities of the joints [Rubrecht et al., 2010].

2.3.4 Contact Constraints

When a robot interacts with its environment, it does so through contacts. These contacts can be *unilateral contacts*, or *bilateral contacts*. Simply put, unilateral contacts are those the robot can only push, e.g. foot contact with the floor, and bilateral contacts are those which allow the robot to push or pull, e.g. gripping the rung of a ladder. For unilateral contact constraints, a linearized approximation of the Coulomb friction cone is employed. Following the formulations in [Salini et al., 2011] and [Saab et al., 2013], a friction contact constraint in the controller must ensure that the linear velocity at the contact point is zero,

$${}^{F}J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + {}^{F}\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} = \mathbf{0}, \tag{2.44}$$

and that the wrench remains within a linearized approximation of a friction cone,

$$^{F}C_i{}^{F}\boldsymbol{\omega}_i \leq \mathbf{0}. \tag{2.45}$$

In (2.44), ^{F}J and $^{F}\dot{J}$ contain the linear components of the i^{th} contact Jacobian. In (2.45), $^{F}C_i$ is a matrix which linearly approximates the second-order norm cone,

$$\left\| ^{F}\boldsymbol{\omega}_i - (^{F}\boldsymbol{\omega}_i \cdot \hat{\boldsymbol{n}}_i)\hat{\boldsymbol{n}}_i \right\|_2 \leq \mu_i(^{F}\boldsymbol{\omega}_i \cdot \hat{\boldsymbol{n}}_i), \tag{2.46}$$

where $^{F}\boldsymbol{\omega}_i$ is are the force components of the i^{th} contact wrench, $\hat{\boldsymbol{n}}_i$ is the normal vector of the contact, and μ_i is the friction coefficient. Finally, expressing these two constraints in terms of $\boldsymbol{\chi}$, and defining $^{F}\boldsymbol{\omega}_i = S_i^F \boldsymbol{\chi}$, gives the following coupled equality and inequality constraints,

$$\underbrace{\begin{bmatrix} ^{F}J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix}}_{A^{\omega}} \boldsymbol{\chi} = \underbrace{-^{F}\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}}_{\boldsymbol{b}^{\omega}} \tag{2.47}$$

$$\underbrace{\begin{bmatrix} \mathbf{0} & ^{F}C_i S_i^F \end{bmatrix}}_{G^{\omega}} \boldsymbol{\chi} \leq \underbrace{\mathbf{0}}_{\boldsymbol{h}^{\omega}}, \tag{2.48}$$

where S_i^F selects the i^{th} contact force vector. Equations (2.47) and (2.48) are valid for a single contact point. For surface contacts, e.g. a foot sole, multiple points on the surface can be used for friction contact constraints — usually the four corners of the foot. Equation (2.47) introduces 3 equality constraints for the linear velocity of the contact point. The number of inequality constraints introduced by (2.48) depends on the number of polygon edges used to approximate the friction cone. Here, 6 edges are used, and because of symmetry, this introduces 3 inequality constraints per contact to the whole-body controller.

For bilateral contacts, it is sufficient to ensure no relative motion between the two links, i and j in contact. It should be noted that here a link can be some part of the environment for which a kinematic model exists. To ensure no motion between the links, the following relationship must be true,

$$(J_i(\boldsymbol{q}) - J_j(\boldsymbol{q}))\,\dot{\boldsymbol{\nu}} + \left(\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu}) - \dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})\right)\boldsymbol{\nu} = \mathbf{0}, \tag{2.49}$$

where $J_i(\boldsymbol{q})$, $\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})$, $J_j(\boldsymbol{q})$, and $\dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})$, are the Jacobians and their derivatives for the i^{th} and j^{th} links respectively. Putting (2.49) in terms of $\boldsymbol{\chi}$ produces,

$$\underbrace{\begin{bmatrix} (J_i(\boldsymbol{q}) - J_j(\boldsymbol{q})) & \mathbf{0} \end{bmatrix}}_{A^{bc}} \boldsymbol{\chi} = \underbrace{-\left(\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu}) - \dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})\right)\boldsymbol{\nu}}_{\boldsymbol{b}^{bc}}. \tag{2.50}$$

2.4 Prioritization Strategies

Up to this point, only one task objective function has been considered in the whole-body controller in (2.1). If multiple task objective functions are combined (using operations that preserve convexity) in the resolution of the control problem, then they can be performed simultaneously. In these cases, it is important to select a strategy for the resolution of the optimization problem. In turn, the strategy determines how tasks interact/interfere with one another. The two prevailing methods for dealing with multiple tasks are hierarchical and weighted prioritization.

2.4.1 Hierarchical Prioritization

In *hierarchical prioritization*, the tasks are organized by order of importance in discrete levels. Each task error is minimized in descending order of its importance and the solution to the optimization problem is then used in the equality constraints for the proceeding optimizations.

Algorithm 1 Hierarchical Multi-Objective Optimization

```
for $i = 1 \dots n_{\text{task}}$ do
    $$\begin{aligned} \boldsymbol{\chi}_i^* = \quad & \underset{\boldsymbol{\chi}}{\arg\min} && f_i^{\text{task}}(\boldsymbol{\chi}) + w_0 f_0^{\text{task}}(\boldsymbol{\chi}) \\ & \text{s.t.} && G\boldsymbol{\chi} \leq \boldsymbol{h} \\ & && A_i\boldsymbol{\chi} = \boldsymbol{b}_i \end{aligned}$$
    $A_{i+1} \leftarrow \begin{bmatrix} A_i \\ E_i \end{bmatrix}$
    $\boldsymbol{b}_{i+1} \leftarrow \begin{bmatrix} \boldsymbol{b}_i \\ \boldsymbol{\chi}_i^* \end{bmatrix}$
    $\boldsymbol{\chi}^* \leftarrow \boldsymbol{\chi}_i^*$
end for
return $\boldsymbol{\chi}^*$
```

Algorithm 1 is tantamount to null-space projection in the dynamic domain; however, inequality constraints can be accounted for. As a note, the regularization term, $w_0 f_0^{\text{task}}(\boldsymbol{x})$, in each optimization cascade serves to remove solution redundancy when the objective function has a null space, but this redundancy is necessary for executing the subsequent tasks. The operation, $A_{i+1} \leftarrow \begin{bmatrix} A_i \\ E_i \end{bmatrix}$, propagates the null space of the objective function, which has just been solved, to the proceeding objective functions through the equality constraint. Treating (2.51) hierarchically has the benefit of strictly ensuring the optimization of one task error over another; however, it makes task transitioning and blending more difficult. Using continuous, or soft, priorities can alleviate some of these issues.

2.4.2 Weighted Prioritization

In multi-objective optimization, task weights dictate where, on the Pareto front of solutions, the QP calculates an optimum. Consequently, the optimum found favors the minimization of tasks with higher weights. This affords a method of prioritization, which ensures that critical tasks, such as those for balance, are preferentially accomplished, in situations where other less-critical tasks, such as a reach, have conflicting optima. However, using continuous priorities between tasks cannot guarantee that the tasks will not interfere with one another.

Algorithm 2 Weighted Whole-Body Controller

```
$$\begin{aligned} \boldsymbol{\chi}^* = \quad & \underset{\boldsymbol{\chi}}{\arg\min} && \sum_{i=1}^{n_{\text{task}}} w_i f_i^{\text{task}}(\boldsymbol{\chi}) + w_0 f_0^{\text{task}}(\boldsymbol{\chi}) \\ & \text{s.t.} && G\boldsymbol{\chi} \leq \boldsymbol{h} \\ & && A\boldsymbol{\chi} = \boldsymbol{b}. \end{aligned} \tag{2.51}$$
return $\boldsymbol{\chi}^*$
```

2.4.3 Hybrid Schemes

It can be seen that the weighted strategy is a subset of the hierarchical strategy, by observing that each level in a hierarchical scheme can be solved as a weighted problem. This *hybrid prioritization strategy* can provide the best of both hierarchical and weighted methods, but at the cost of increase implementation and computational complexity. In addition to the simple mixing of weights and hierarchies, continuous generalized projection schemes are developed by [Liu et al., 2016]. These methods allow priorities to continuously vary from weighted to purely hierarchical through scalar values. Such approaches can provide smooth transitions between tasks, as is common in complex activities such as walking, but require substantially more computation time than purely weighted or hierarchical methods.

2.5 Task Servoing

The desired terms, $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$, $\dot{\boldsymbol{\nu}}_i^{\text{des}}$, ${}^{e}\boldsymbol{\omega}_i^{\text{des}}$, and $\boldsymbol{\tau}^{\text{des}}$, from (2.10), (2.16), (2.20), and (2.27), respectively are provided by higher-level task servoing. Commonly, the high-level reference of a task is simply to attain some pose, and in the case of a wrench task, some force and/or torque. For acceleration tasks, if the desired task value is expressed as a pose, position, or orientation, then it must be converted to an acceleration. This is done here using a feedforward (PD) controller,

$$\ddot{\boldsymbol{\xi}}_i^{\text{des}}(t+\Delta t) = \ddot{\boldsymbol{\xi}}_i^{\text{ref}}(t+\Delta t) + K_p \boldsymbol{\epsilon}_i(t) + K_d \dot{\boldsymbol{\epsilon}}_i(t), \tag{2.52}$$

where $\ddot{\boldsymbol{\xi}}_i^{\text{ref}}(t+\Delta t)$ is the feedforward frame acceleration term, $\boldsymbol{\epsilon}_i(t)$ and $\dot{\boldsymbol{\epsilon}}_i(t)$ are the current pose error and its derivative, with K_p and $K_d = 2\sqrt{K_p}$, their proportional and derivative gains respectively. As seen in Sec. 2.2, this term also serves to remove drift at the controller level and stabilize the output of the task. The terms, $\boldsymbol{\epsilon}_i(t)$ and $\dot{\boldsymbol{\epsilon}}_i(t)$, are not explicitly defined here because they are representation dependent (see [Siciliano and Khatib, 2008]). For wrench and torque tasks a similar servoing controller can be developed using a Proportional-Integral (PI) controller.

$$\boldsymbol{\omega}^{des}(t+\Delta t) = \boldsymbol{\omega}^{ref}(t+\Delta t) + K_p \boldsymbol{\epsilon}_{\boldsymbol{\omega}}(t) + K_i \int \boldsymbol{\epsilon}_{\boldsymbol{\omega}}(t)dt \tag{2.53}$$

This servoing helps stabilize the whole-body controller by driving the desired task values to some fixed state in asymptotically stable manner. Without the servoing the the task error objective term, $f_i^{\text{task}}(\boldsymbol{\chi})$, could change discontinuously between time steps resulting in discontinuous jumps in the optimal joint torques determined between two time steps. See Sec. 1.1.2 for more about task servoing.

2.6 Conclusions

The equations and relationships presented in this chapter are sufficient for understanding the methods presented in this study. However, they are far from covering the entire body of work on reactive whole-body control. There are many ways to formulate whole-body controllers, each with its own advantages and disadvantages, and a review of these controllers is outside the scope of this study. The critical information to retain from this chapter is the fact that a controller can basically be separated into two parts, based on the fact that it is simply an optimization problem. The first part consists of the objective function(s), which are the control tasks being performed. These tasks are designed to produce some desired overall behavior and may or may not be achievable. They are *objectives* the robot operator, or higher-level planning, seeks to accomplish with varying degrees of importance, and are in no way guaranteed to be carried out by the robot. The second part consists of the control constraints. These constraints dictate what the robot can or cannot do, and must be respected absolutely. This distinction between objectives and constraints is an important concept which lays the groundwork for understanding the concepts of task compatibility and feasibility in the following chapter.

Part III

METHODS

Chapter 3

Mathematical Foundations of Task Compatibility and Feasibility

In this chapter, we try to understand mathematically what makes tasks incompatible or infeasible. To do so, we explore the properties of the underlying multi-objective convex optimization problem within the whole-body controller described in Chapter 2. This allows us to formulate a series of metrics and measures based only on the details of numerical optimization and linear systems of equations. These metrics quantify the compatibility and feasibility of objective functions (i.e. tasks).

3.1 Introduction

When multiple simultaneous tasks are executed on a robot, it is often the case that the tasks interfere with one another. An example of this would be a postural task preventing an end-effector Cartesian task from perfectly following its reference trajectory. The big question then, is why exactly does this happen? From a roboticist's perspective, this sort of interference could be explained by any number of physical phenomena common to robotics: a loss of redundancy, hitting joint limits, actuator limits, limited workspace, etc. Numerous indicators have been developed to try to capture various combinations of these phenomena (see Chapter 1). However, these root causes may be difficult to identify for a given set of tasks without closely studying the problem. Additionally, these causes may only occur at certain instants in the movement.

Looking at how the control problem is formulated, we can gain insight into the numerical causes of these problems. To be more specific, the model-based whole-body controller presented in Chapter 2 is formulated as a constrained convex optimization problem. Within the optimization problem are the control objectives and constraints. Therefore, virtually all possible reasons for tasks interfering with one another are embedded in the control problem as either objectives or constraints. The objectives (tasks) are either incompatible with one another, and/or infeasible with the constraints. It should be noted that in this work, the case of unknown/un-modeled perturbations occurring is ignored. Such scenarios introduce infinitely many possible causes for task incompatibility and infeasibility and are considered another challenge entirely, as they require high-level decision making to replan the whole-body behavior; thus, they are outside the scope of this work.

In this chapter, a mathematical formalism is developed for the notions of task compatibility and feasibility. Following the work of [Bouyarmane and Kheddar, 2015] and [Wieber et al., 2017], this is accomplished by looking at the problem of task compatibility and feasibility from a numerical optimization perspective rather than a control perspective. Therefore, the notion of a task is put aside for the moment, and this chapter focuses on the more generic concept of *objective compatibility and feasibility.*

3.2 A Brief Overview of Convex Optimization

In this section, the fundamental concepts of convex optimization, and specifically linearly constrained Quadratic Programs (QPs), are succinctly presented. It begins with a general overview of what constitutes a convex optimization problem and then moves on to the specifics of QPs. Finally, multi-objective QPs are presented.

3.2.1 A General Convex Problem

An optimization problem is generically formulated as the minimization (or maximization) of a function, subject to any number of equality and inequality constraints,

$$\begin{aligned} \underset{\boldsymbol{x}}{\arg\min} \quad & f(\boldsymbol{x}) && (3.1)\\ \text{s.t.} \quad & b(\boldsymbol{x}) = \mathbf{0} && (3.2)\\ & g(\boldsymbol{x}) \leq \mathbf{0}. && (3.3) \end{aligned}$$

This work focuses on continuous optimization, where $\boldsymbol{x} \in \mathbb{R}^n$. The function, $f(\boldsymbol{x})$, is the problem *objective*, or objective function. The variable which is used to minimize $f(\boldsymbol{x})$ is $\boldsymbol{x}$ and is known as the *optimization variable*. This minimization may be subject to equality, (3.2), and/or inequality constraints, (3.3), on the optimization variable. In the general non-linear and non-convex case, (3.1) may admit zero or infinitely many optima making it difficult or impossible to efficiently find a global minimum. In the special case where,

$$f(\theta \boldsymbol{a} + (1-\theta)\boldsymbol{b}) \leq \theta f(\boldsymbol{a}) + (1-\theta) f(\boldsymbol{b}), \tag{3.4}$$

for all $\boldsymbol{a}$ and $\boldsymbol{b}$ in the domain of $f(\boldsymbol{x})$, and $0 \leq \theta \leq 1$, the function $f(\boldsymbol{x})$ is convex (or even affine). Figure 3.1 shows two examples of such functions. If the equality and inequality constraints are affine or quadratic in $\boldsymbol{x}$ as well, then (3.1) is convex in $\boldsymbol{x}$ and can be solved globally and efficiently. The sets of problems that fall into this category are known as *convex optimization* problems. The reader is directed to [Boyd and Vandenberghe, 2004] for more details on convex optimization. When $f(\boldsymbol{x})$ is quadratic in $\boldsymbol{x}$ and the constraints are affine in $\boldsymbol{x}$, the problem becomes a QP, which is a special subset of convex optimization that is prevalent in optimal control problems.

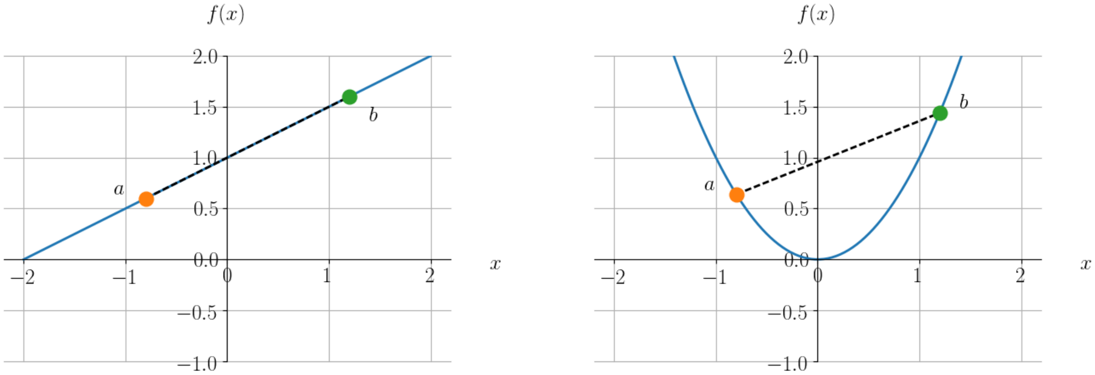
3.2.2 Quadratic Programming

Quadratic Programs (QPs) derive their name from having a quadratic objective function which is expressed as,

$$\begin{aligned} \underset{\boldsymbol{x}}{\arg\min} \quad f(\boldsymbol{x}) &= \frac{1}{2}\boldsymbol{x}^\top P \boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r && (3.5)\\ &= \boldsymbol{x}^\top (E^\top E)\boldsymbol{x} - 2(E^\top \mathbf{f})^\top \boldsymbol{x} + \mathbf{f}^\top \mathbf{f} &&\\ &= \left\| E\boldsymbol{x} - \mathbf{f} \right\|_2^2, && (3.6) \end{aligned}$$

where (3.5) is the classical formulation of a QP, and (3.6) is the least-squares formulation. We will continue using (3.6), which admits an analytical minimum-norm solution, $\boldsymbol{x}^*$, in the unconstrained case.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\arg\min} \ \left\| E\boldsymbol{x} - \mathbf{f} \right\|_2^2 = E^\dagger \mathbf{f}, \tag{3.7}$$



(a) an affine function, $f(x) = 0.5x + 1$

(b) a quadratic function, $f(x) = x^2$

Figure 3.1 – Two examples of convex functions. For each example, two test points, a and b, are chosen in the domain of $x \in \mathbb{R}^1$ to show that (3.4) is respected. This can be seen by observing that the chord which connects the points a and b lies on or within the function epigraphs, which are the sets of all points which lie above the function graphs.

where $E^\dagger$ is the Moore-Penrose pseudoinverse of the E matrix. This solution will be found assuming the rank of the linear system is consistent. Details on rank conditions are presented in depth in Sec. 3.3.1. Adding an affine equality constraint to (3.6) produces a constrained least squares problem,

$$\begin{aligned} \underset{\boldsymbol{x}}{\arg\min} \quad & \|E\boldsymbol{x} - \mathbf{f}\|_2^2 && (3.8) \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b}, && (3.9) \end{aligned}$$

which can be solved analytically, assuming a solution exists, using the *Karush Kuhn Tucker (KKT) equations*,

$$\underbrace{\begin{bmatrix} E^\top E & A^\top \\ A & \mathbf{0} \end{bmatrix}}_{\text{KKT Matrix}} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} E^\top \mathbf{f} \\ \boldsymbol{b} \end{bmatrix} \tag{3.10}$$

$$\Leftrightarrow \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} E^\top E & A^\top \\ A & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} E^\top \mathbf{f} \\ \boldsymbol{b} \end{bmatrix}, \tag{3.11}$$

where $\boldsymbol{z}$ is the solution to the dual problem and contains the *Lagrange multipliers*. Adding an affine inequality constraint to the problem produces the following QP,

$$\begin{aligned} \underset{\boldsymbol{x}}{\arg\min} \quad & \|E\boldsymbol{x} - \mathbf{f}\|_2^2 && (3.12) \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b} && (3.13) \\ & G\boldsymbol{x} \leq \boldsymbol{h}. && (3.14) \end{aligned}$$

Equation (3.12) can no longer be solved analytically and one must use numerical methods such as interior point, or active set methods [Boyd and Vandenberghe, 2004]. Resolution of (3.12) will provide a globally optimal solution for $\boldsymbol{x}^*$ provided that the constraint equations are consistent, i.e. the set of possible solutions is not empty.

3.2.3 Multi-Objective Optimization

Objective functions represent the intentions of the problem designer: what meaningful quantity or measure is to be minimized to best solve some issue. As is often the case, there may be more than one quantity or measure which must be minimized and therefore multiple objective functions are combined together. When multiple objective functions, $f_i(\boldsymbol{x})$, are considered simultaneously, a *multi-objective optimization* problem (a.k.a. multicriteria, multicriterion, or Pareto optimization) is created. One common method of solving multi-objective optimization problems is through *scalarization*. Scalarization is the process of combining of multiple objective costs into one scalar cost. There are a multitude of scalarization techniques but weighted summation is of the most common,

$$\arg\min_{\boldsymbol{x}} \sum_{i=1}^{n_o} w_i f_i(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \left\| E_i \boldsymbol{x} - \mathbf{f}_i \right\|_2^2 . \tag{3.15}$$

In (3.15), n_o is the total number of objective functions. This scalarization can be written compactly by concatenating the individual objectives as,

$$\arg\min_{\boldsymbol{x}} \quad \left\| E_w \boldsymbol{x} - \mathbf{f}_w \right\|_2^2 \tag{3.16}$$

where

$$E_w = \begin{bmatrix} \sqrt{w_1} E_1 \\ \sqrt{w_2} E_2 \\ \vdots \\ \sqrt{w_n} E_{n_o} \end{bmatrix} \quad \text{and} \quad \mathbf{f}_w = \begin{bmatrix} \sqrt{w_1} \mathbf{f}_1 \\ \sqrt{w_2} \mathbf{f}_2 \\ \vdots \\ \sqrt{w_n} \mathbf{f}_{n_o} \end{bmatrix} . \tag{3.17}$$

Each weight, $w_i \geq 0$, dictates the relative importance of its objective $f_i(\boldsymbol{x})$ and therefore its impact on the solution. In (3.16) the weights are assumed to be scalars, but it is also possible to use matrices of different weights as long as they remain positive semi-definite.

As an alternative to scalarization, the objective functions can be minimized hierarchically in order of importance to ensure that the most important objective(s) are minimized as much as possible without influence of the lower priority objectives. This is known as *lexicographic optimization* in multi-objective optimization. To achieve this, the objectives are treated individually as a cascade of QPs where the solutions are reused as equality constraints in the subsequent QP minimizations. See Algorithm 1 from Sec. 2.4.1 for an implementation outline of hierarchical optimization.

3.2.4 Comparing Weighted Scalarization and Hierarchical Optimization

To understand the impact of both weighted scalarization and hierarchical optimization on the overall solution of the multi-objective problem, we take the following two dimensional problem, i.e. $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top$, with 4 trivial objective functions.

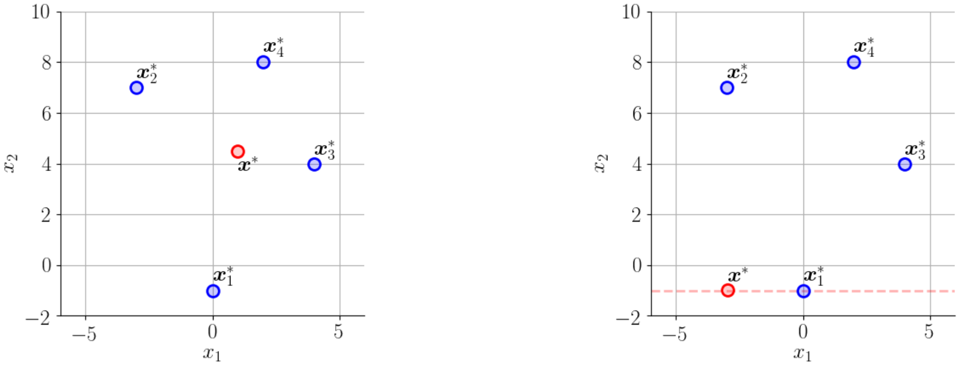
$$f_i(\boldsymbol{x}) = w_i \left\| E_i \boldsymbol{x} - \mathbf{f}_i \right\|_2^2 + w_0 \left\| \boldsymbol{x} \right\|_2^2 \quad \forall i \in 1 \dots 4, \tag{3.18}$$

where $\left\| \boldsymbol{x} \right\|_2^2$ is the so-called regularization term, which ensures a solution for problems with infinitely many solutions, and $0 \leq w_0 \ll 1$, is a small positive weight; here, $w_0 = 1e - 10$. The effects of the objective rank are explored in detail in Sec. 3.3.1. The chosen numerical values are as follows,

$$E_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad E_2 = E_3 = E_4 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.19}$$

$$\mathbf{f}_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} -3 \\ 7 \end{bmatrix} \quad \mathbf{f}_3 = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad \mathbf{f}_4 = \begin{bmatrix} 2 \\ 8 \end{bmatrix} . \tag{3.20}$$

By choice E_1 is rank 1 and therefore has a nullity of 1 as well. This means any value of x_1 can satisfy the minimization problem, and that a regularization term must be included in order to ensure convergence of the optimization. The unconstrained optimal value of all four objective functions can be easily determined as $x_i^* = \mathbf{f}_i$. Given the following objective function weights, $w_1 = w_2 = w_3 = w_4 = 1$, one obtains an overall optimum at $\boldsymbol{x}^* = \begin{bmatrix}1.0 & 4.5\end{bmatrix}^\top$ using weighted sum scalarization. Using the objective function numbering as a prioritization order, with 1 being the highest priority, a hierarchical optimization provides an overall optimum at $\boldsymbol{x}^* = \begin{bmatrix}-3.0 & -1\end{bmatrix}^\top$. Both results are shown in Fig. 3.2. In Fig. 3.2a, the weighted case, it can be seen that $\boldsymbol{x}^*$ is a mix of all of the individual objective optima, $\boldsymbol{x}_i^*$, with each objective imparting equal influence on the overall optimum. In the hierarchical case, the minimization of $f_1(\boldsymbol{x})$ dictates that any proceeding solutions lie within its null space, indicated by the red dashed line in Fig. 3.2b. The next objective to be minimized is $f_2(\boldsymbol{x})$ and only the x_1 component of its solution is compatible. At this point, the equality constraints defined by the first two levels of the QP hierarchy fully specify a solution and the optimization is stopped.



(a) weighted sum scalarization

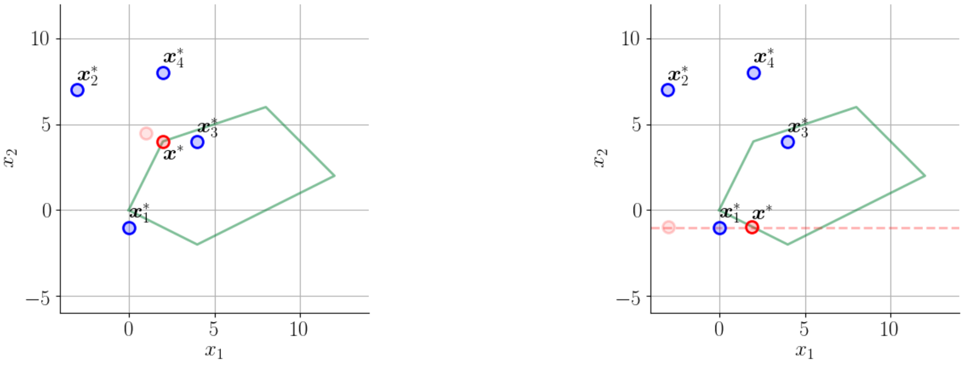
(b) hierarchical optimization

Figure 3.2 – Examples of unconstrained optima, $\boldsymbol{x}^*$, found using both weighted scalarization and hierarchical optimization. The individual unconstrained objective optima, $\boldsymbol{x}_i^*$, are shown as the blue dots. For the weighted case, (a), all weights are equal to 1.0. In the hierarchical case, (b), the numbering of the objective functions dictates the objective priority, with $i = 1$ being of highest priority.

Regardless of the method, when dealing with multiple objective functions, compromises must be made between them, except in the trivial case where all objective functions share the same global optimum. Things are further complicated by the introduction of constraints which can limit the feasible set. Introducing the following inequality constraints,

$$\underbrace{\begin{bmatrix}-0.9 & 0.4\\ -0.3 & 0.9\\ 0.7 & 0.7\\ 0.4 & -0.9\\ -0.4 & -0.9\end{bmatrix}}_{G}\begin{bmatrix}x_1\\ x_2\end{bmatrix} \leq \underbrace{\begin{bmatrix}0\\ 3.2\\ 9.9\\ 3.6\\ 0\end{bmatrix}}_{\boldsymbol{h}}, \tag{3.21}$$

produces the following global optima, $\boldsymbol{x}^* = \begin{bmatrix}2.0 & 4.0\end{bmatrix}^\top$ and $\boldsymbol{x}^* = \begin{bmatrix}1.9 & -1.0\end{bmatrix}^\top$, for the weighted and hierarchical cases respectively. Figure 3.3 shows the constraint polygon and optimization results graphically. In the weighted case, 3.3a, $\boldsymbol{x}^*$ falls on the vertex of two constraints which is the closest point in the feasible set to the unconstrained optimum shown in transparent red. For the hierarchical optimization, 3.3b, the first objective function minimization is constrained by an inequality and this immediately restricts the solution space, precluding further optimization of the objective functions.



(a) weighted sum scalarization

(b) hierarchical optimization

Figure 3.3 – Examples of constrained optima, $\boldsymbol{x}^*$, found using both weighted scalarization and hierarchical optimization. The individual unconstrained objective optima, $\boldsymbol{x}_i^* \ \forall i \in 1 \dots 4$, are shown as the blue dots. The unconstrained optima found using both methods are shown in translucent red.

3.3 Objective Compatibility and Feasibility

With the basic concepts of convex multi-objective optimization presented in Sec. 3.2, two crucial notions are formalized: *objective compatibility* and *objective feasibility*. The concept of compatibility indicates how well the various objective functions can be minimized when mixed together using weighted scalarization or hierarchical optimization. Objective feasibility gives a notion of how well the individual and combined objective functions respect the problem constraints, or conversely, how much the constraints restrict their minimization. In the following section, a formal definition of these concepts is built in the context of convex multi-objective optimization.

3.3.1 Objective Compatibility

The goal of this section is to develop a concrete definition of what it means for two or more objective functions to be compatible with one another. To do so, we start by contradiction and discuss what it means for objectives to be *incompatible*.

Strict Objective Compatibility

If two or more objective functions are combined in a multi-objective optimization problem, using weighted scalarization or hierarchical optimization, and do not share the same global optimum, then some compromise is necessarily made between them in the solution of the multi-objective problem. To be explicit, compromise means that one or more objective functions will not be perfectly minimized, i.e.

$$\nabla f_i(\boldsymbol{x}^*) \neq 0. \tag{3.22}$$

For a QP this can be written as,

$$\left\| \boldsymbol{x}^* - E_i^{\dagger} \mathbf{f}_i \right\|_2 > 0 \tag{3.23}$$

which says that if the Euclidean norm of the distance from $\boldsymbol{x}^*$ to the individual unconstrained objective optimum, $\boldsymbol{x}_i^* = E_i^\dagger \mathbf{f}_i$, is greater than 0 then one or more of the individual objective functions is incompatible with the others. A first trivial definition of objective compatibility could then be formulated as,

$$E_i \boldsymbol{x}^* = \mathbf{f}_i \quad \forall i \in 1 \dots n_o, \tag{3.24}$$

which basically comes down to the question of whether or not there exists an $\boldsymbol{x}^*$ which perfectly satisfies,

$$\tilde{E} \boldsymbol{x}^* = \tilde{\mathbf{f}}, \tag{3.25}$$

where $\tilde{E}$ and $\tilde{\mathbf{f}}$ are defined by the concatenation of the n_o least-squares objective functions,

$$\tilde{E} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n_o} \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{f}} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_o} \end{bmatrix}. \tag{3.26}$$

It can therefore be stated that $\exists\ \boldsymbol{x}^* : \tilde{E}\boldsymbol{x}^* = \tilde{\mathbf{f}}$ if,

$$\operatorname{rank}(\tilde{E}) = \operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}]) \leq n, \tag{3.27}$$

where $[\tilde{E}|\tilde{\mathbf{f}}]$ is the augmented matrix of the linear system of equations, and n is the number of columns in $\tilde{E}$, or alternatively, the number of rows in $\boldsymbol{x}$. If,

$$\operatorname{rank}(\tilde{E}) < \operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}]), \tag{3.28}$$

then the system is *inconsistent* and no solution exists. If,

$$\operatorname{rank}(\tilde{E}) = \operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}]) = n, \tag{3.29}$$

then there is exactly one solution which satisfies (3.25), while if

$$\operatorname{rank}(\tilde{E}) = \operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}]) < n, \tag{3.30}$$

then there exist infinitely many solutions. It must also be noted, that these rank conditions are also valid for one objective, not just a combination of objectives. In this case the issue is whether or not the objective function is consistent and not compatible.

Example 1

Given two objective functions, we wish to determine if they are strictly compatible.

$$f_1 = \|E_1\boldsymbol{x} - \mathbf{f}_1\| \tag{3.31}$$
$$f_2 = \|E_2\boldsymbol{x} - \mathbf{f}_2\| \tag{3.32}$$

CASE 1:
In this first case the following numerical values are used:

$$E_1 = \begin{bmatrix} 1.2 & 3.0 & -1.0 \\ 2.3 & -1.5 & 4.0 \end{bmatrix} \quad \mathbf{f}_1 = \begin{bmatrix} 2.92 \\ 1.67 \end{bmatrix} \tag{3.33}$$
$$E_2 = \begin{bmatrix} -3.2 & 1.7 & -2.4 \\ 2.0 & 3.1 & 1.5 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} -2.818 \\ 4.876 \end{bmatrix}. \tag{3.34}$$

Evaluating (3.27) for case 1,

$$\operatorname{rank}\left(\begin{bmatrix} E_1 \\ E_2 \end{bmatrix}\right) = 3 = \operatorname{rank}\left(\left[\begin{array}{c|c} E_1 & \mathbf{f}_1 \\ E_2 & \mathbf{f}_2 \end{array}\right]\right) = \operatorname{rows}(\boldsymbol{x}), \tag{3.35}$$

it can be determined that there is exactly one $\boldsymbol{x}_1^* = \tilde{E}^\dagger\tilde{\mathbf{f}}$ which can perfectly satisfy both objective functions. In this case, $\boldsymbol{x}_1^* = \begin{bmatrix} 1.2 & 0.46 & -0.1 \end{bmatrix}^\top$.

CASE 2:
In this case the second row of each E and $\mathbf{f}$ from case 1 is removed,

$$E_1 = \begin{bmatrix} 1.2 & 3.0 & -1.0 \end{bmatrix} \quad \mathbf{f}_1 = \begin{bmatrix} 2.92 \end{bmatrix} \tag{3.36}$$
$$E_2 = \begin{bmatrix} -3.2 & 1.7 & -2.4 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} -2.818 \end{bmatrix}, \tag{3.37}$$

then,

$$\operatorname{rank}\left(\begin{bmatrix} E_1 \\ E_2 \end{bmatrix}\right) = 2 = \operatorname{rank}\left(\left[\begin{array}{c|c} E_1 & \mathbf{f}_1 \\ E_2 & \mathbf{f}_2 \end{array}\right]\right) < \operatorname{rows}(\boldsymbol{x}). \tag{3.38}$$

In (3.38), it can be observed that, because the rank of the linear system is less than the number of problem variables, there will be infinitely many solutions which satisfy the system. In this case, the solution from (3.35) remains valid, i.e. $\tilde{E}\boldsymbol{x}_1^* = \tilde{\mathbf{f}}$, but the solution found using the pseudoinverse, $\boldsymbol{x}_2^* = \tilde{E}^\dagger\tilde{\mathbf{f}} = \begin{bmatrix} 1.0652 & 0.609 & 0.1852 \end{bmatrix}^\top$ is different than before because it minimizes the norm of $\boldsymbol{x}^*$ as well,

$$\|\boldsymbol{x}_2^*\|_2 < \|\boldsymbol{x}_1^*\|_2 \tag{3.39}$$
$$\Rightarrow 1.24 < 1.29. \tag{3.40}$$

CASE 3:
Finally, as a counterexample, we take the values used in (3.33) and (3.34) from case 1, and slightly alter $\mathbf{f}_2$,

$$\mathbf{f}_2 = \begin{bmatrix} -2.818 \\ 1 \end{bmatrix}, \tag{3.41}$$

and evaluate the rank conditions,

$$\operatorname{rank}\left(\begin{bmatrix} E_1 \\ E_2 \end{bmatrix}\right) = 3 < \operatorname{rank}\left(\left[\begin{array}{c|c} E_1 & \mathbf{f}_1 \\ E_2 & \mathbf{f}_2 \end{array}\right]\right) = 4, \tag{3.42}$$

which imply that no solution, $\boldsymbol{x}_3^*$, exists which satisfies the two objective functions.

For the first two cases studied in Example 1, the solutions found, $\boldsymbol{x}_1^*$ and $\boldsymbol{x}_2^*$, hold regardless of whether weighted scalarization or hierarchical optimization is used. While in the third case it is certain that no $\boldsymbol{x}_3^*$ can be found which perfectly minimizes the two objective functions. A strict definition of objective compatibility can therefore be written as,

> **Strict objective compatibility**: A set of objective functions are compatible if there exists an $\boldsymbol{x}^* \in \mathbb{R}^n$ which satisfies, $\tilde{E}\boldsymbol{x}^* = \tilde{\mathbf{f}}$, and consequently $\text{rank}(\tilde{E}) = \text{rank}([\tilde{E}|\tilde{\mathbf{f}}]) \leq n$, where $\tilde{E}$ and $\tilde{\mathbf{f}}$ are written as in (3.26).

Applying this definition to the chapter example from (3.19) and (3.20), one obtains the relatively evident solution of $\text{rank}(\tilde{E}) = 2 < \text{rank}([\tilde{E}|\tilde{\mathbf{f}}]) = 3$, which indicates that no solution $\boldsymbol{x}^*$ can satisfy all objective functions. These rank conditions establish a binary criterion for objective compatibility, and open the question of how to measure incompatibility for inconsistent cases such as the one observed in (3.42).

Objective Compatibility Metrics

While precise, imposing that all tasks be perfectly compatible, even for high-dimensional problems, is overly restrictive and probably impossible in most cases. It is therefore necessary to develop measures of compatibility which can provide finer gradations than simply, compatible or incompatible. A possible starting point for such a measure is to approximate the rank conditions for strict compatibility on a continuous scale. As proven by [Recht et al., 2010], the *nuclear norm*, or *trace norm*, can be used as an approximation[1] of the rank of a matrix, $M \in \mathbb{R}^{m\times n}$, and is defined by the sum of its singular values, σ_i,

$$\|M\|_* := \sum_{i=1}^{n} \sigma_i(M). \tag{3.43}$$

Given that strict compatibility represents the state of maximal compatibility, and is defined when $\text{rank}(\tilde{E}) = \text{rank}([\tilde{E}|\tilde{\mathbf{f}}])$, a useful application of the nuclear norm is to approximate how far from the strict compatibility criterion the objectives are. This is possible by defining the *Nuclear norm Ratio (NR)*,

$$\kappa^{\text{NR}} = \frac{\left\|\tilde{E}\right\|_*}{\left\|\left[\tilde{E}|\tilde{\mathbf{f}}\right]\right\|_*}. \tag{3.44}$$

This ratio is bounded between $[0, 1]$. As κ^{NR} tends towards one, the objectives should approach strict compatibility, while a κ^{NR} tending towards zero would indicate decreasing compatibility.

Moving away from rank-based criteria, the distance between each objective's unconstrained optimum and the multi-objective optimum, or *Optimum Distance* (OD) can be determined,

$$\kappa_i^{\text{OD}} = \|\boldsymbol{x}^* - \boldsymbol{x}_i^*\|_2\,. \tag{3.45}$$

In (3.45), κ_i^{OD} is the compatibility of the i^{th} objective function w.r.t. the chosen global optimum $\boldsymbol{x}^*$, and therefore, w.r.t. the other objective functions. Summing over the n objective functions, the κ^{OD} metric can be defined as,

$$\kappa^{\text{OD}} = \sum_{i=1}^{n} \kappa_i^{\text{OD}}. \tag{3.46}$$

Another approach, similar to the norm metrics of compatibility, is to look directly at the individual objective functions and evaluate them at $\boldsymbol{x}^*$ giving the *Optimum Cost* (OC) metric,

$$\kappa_i^{\text{OC}} = w_i f_i(\boldsymbol{x}^*)\ \forall i \in 1 \ldots n. \tag{3.47}$$

---

[1] Some bounds on the error of this approximation can be found in [Srebro and Shraibman, 2005], and parallels can be drawn with trace minimization heuristics in automatic control as in [Mesbahi and Papavassilopoulos, 1997]. A good explanation of the nuclear norm can be found in [Osnaga, 2014].

In (3.47), w_i, is the weight of f_i, if weighted scalarization is used. If hierarchical scalarization is used then one can either omit weights in (3.47), or use approximate weights which would lead to nearly hierarchical resolution of the QP; e.g. $w_1 = 1.0$, $w_2 = 1.0e-5$, $w_3 = 1.0e-10$, etc. The objectives with the largest κ_i^{OC} are those which are most poorly minimized and therefore the least compatible with the others. Looking at the example problem, in Table 3.1, it can be observed that objective f_1 has the highest κ^{OC} value for the weighted case and the lowest for the hierarchical case, which follows logically in that it is the highest priority objective in the latter method. Using these measures, one can conclude which objectives are the most and least compatible by observing those which are most and least minimized, respectively. Additionally, it can be shown that different prioritization strategies yield different overall compatibilities, by looking at the sum of the κ_i^{OC},

$$\kappa^{\mathrm{OC}} = \sum_{i=1}^{n} \kappa_i^{\mathrm{OC}}. \tag{3.48}$$

Again, in Table 3.1, the results of (3.48) for each prioritization strategy are shown, and it can be concluded that the resulting $\boldsymbol{x}^*$ from the weighted scalarization produces the most compatible overall solution, in the sense that the overall sum cost of the individual objectives is less than that of the hierarchical case.

prioritization strategy	κ_1^{OC}	κ_2^{OC}	κ_3^{OC}	κ_4^{OC}	κ^{OC}
weighted	30.25	22.25	9.25	13.25	75
hierarchical	$9.8e-05$	63.8	73.5	105.5	243

Table 3.1 – Computation of the objective compatibility measure defined in (3.47) for the $\boldsymbol{x}^*$ calculated using weighted scalarization and hierarchical optimization — both in the unconstrained case.

Compatibility Ellipsoid

Alternatively, one could look for more global descriptors of objective compatibility by looking at the objectives as a whole rather than individually. Each objective has an unconstrained optimum, $\boldsymbol{x}_i^* = E_i^{\dagger}\mathrm{f}_i$, and the finite set of these optima can be written,

$$\mathcal{C} = \{\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_n^*\}. \tag{3.49}$$

With the set $\mathcal{C}$, the minimum volume ellipsoid, which covers the convex hull defined by $\mathcal{C}$, can be written

$$\mathcal{E}_\kappa = \{\boldsymbol{u} \mid \|A\boldsymbol{u} + \boldsymbol{b}\|_2 \leq 1\}, \tag{3.50}$$

with A a square symmetric positive definite matrix. This is known as the *Löwner John Ellipsoid*. The A and $\boldsymbol{b}$ matrices can be determined by solving the following convex optimization problem,

$$\begin{aligned} \arg\min_{A,\boldsymbol{b}} \quad & \log\det A^{-1} \\ \text{s.t.} \quad & \|A\boldsymbol{x}_i^* + \boldsymbol{b}\|_2 \leq 1 \ \forall i \in 1 \ldots n. \end{aligned} \tag{3.51}$$

To solve (3.51), the log det A^{-1} term is transformed into $\det(A^{\frac{1}{n}})$, where n is again the dimension (number of rows) of $\boldsymbol{x}$, and a *Semidefinite Programming* (SDP) solver can be used. See p. 411 of [Boyd and Vandenberghe, 2004] for more details. This ellipsoid covers the volume of the convex set defined by $\mathcal{C}$. Figure 3.4 shows the Löwner John Ellipsoid for the example problem from Sec. 3.2.4. Equation (3.51) defines the inverse image of a Euclidean unit ball under an affine mapping. This formulation can be converted to the more common representation of an ellipsoid,

$$\mathcal{E}_\kappa = \{B_\kappa \boldsymbol{w} + \boldsymbol{c}_\kappa \mid \|\boldsymbol{w}\| \leq 1\}, \tag{3.52}$$

with the following relationships:

$$\boldsymbol{c}_\kappa = -A^{-1}\boldsymbol{b} \tag{3.53}$$

$$B_\kappa = -A^{-1}. \tag{3.54}$$

Here, $\boldsymbol{c}_\kappa$ is the center of the ellipsoid and the eigenvectors and eigenvalues of B_κ determine the directions and magnitudes of the semi-principal axes respectively. Using Singular Value Decomposition (SVD), these values can be determined easily since B_κ is square symmetric,

$$B_\kappa = USV^\top \tag{3.55}$$

$$U = \begin{bmatrix}\boldsymbol{u}_1 & \boldsymbol{u}_2 & \dots & \boldsymbol{u}_n\end{bmatrix} \tag{3.56}$$

$$S = \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_n), \tag{3.57}$$

where $\boldsymbol{u}_i$ are the eigenvectors and λ_i are the eigenvalues which represent the direction and length, $\sigma_i = \lambda_i$, of each semi-axis of the ellipsoid. Again, n is the dimension of the optimization variable $\boldsymbol{x}$.

From a compatibility standpoint, this ellipsoid can compactly approximate the directions of greatest and least objective compatibility. This ellipsoid is therefore denoted the *compatibility ellipsoid*. Those directions with the largest eigenvectors represent the directions of least compatibility in the optimization variable space. If a principal axis becomes degenerate, i.e. $\lambda_i \simeq 0$, then the objective functions are maximally compatible in that direction. For the example problem from Sec. 3.2.4,

$$S = \begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix} = \begin{bmatrix}5.2 & 0\\ 0 & 3.6\end{bmatrix}, \tag{3.58}$$

These singular values indicate that the majority of the incompatibility is in the σ_1 principal axis and by looking at the corresponding $\boldsymbol{u}_1$ vector's components in the x_1 and x_2 directions,

$$U = \left[\begin{array}{cc} -0.25 & 0.97 \\ \underbrace{0.97}_{\boldsymbol{u}_1} & \underbrace{0.25}_{\boldsymbol{u}_2} \end{array}\right] \tag{3.59}$$

it can be seen that the x_2 dimension has the greatest amount of incompatibility. This can be confirmed visually in Fig. 3.4.

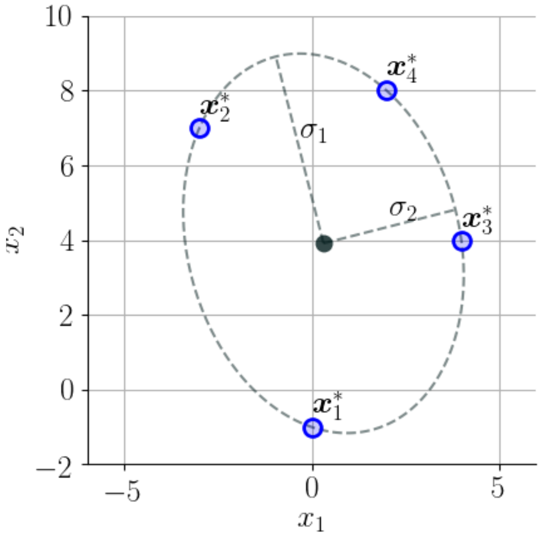


Figure 3.4 – The compatibility ellipsoid, or Löwner John ellipsoid over the objective optima, $\boldsymbol{x}_i^*$, for the example problem from Sec. 3.2.4

The compatibility ellipsoid indicates not only the directions of greatest or least compatibility, but also provides a global measure of compatibility through its *Ellipsoid Volume* (EV), i.e.

$$\kappa^{\text{EV}} = \det(B_\kappa) = \prod_{i=1}^{n} \sigma_i. \tag{3.60}$$

A perfectly compatible set of objectives would result in a completely degenerate ellipse with zero volume, so any non-zero volume should be an indicator of some objective incompatibility. In the case of our example shown in Fig. 3.4, $\kappa^{\text{EV}} = 18.75$. For perfectly compatible objectives, however, the compatibility ellipsoid cannot be computed because (3.51) becomes infeasible for $A \preceq 0$, which occurs when the volume of the ellipsoid goes to zero. The chosen SDP solver should then indicate the infeasibility of the problem and this could be used as an indicator of compatibility.

One final option for measuring compatibility is to use the compatibility ellipsoid center in the place of $\boldsymbol{x}^*$ in (3.46) and calculate an *Ellipsoid Distance* (ED) metric,

$$\kappa^{\mathrm{ED}} = \sum_{i=1}^{n_o} \kappa_i^{\mathrm{ED}} = \sum_{i=1}^{n_o} \left\| \boldsymbol{c}_\kappa - \boldsymbol{x}_i^* \right\|_2 . \tag{3.61}$$

One advantage of this approach is that it is independent of the scalarization technique used to solve the optimization problem.

3.3.2 Objective Feasibility

While objective compatibility captures the relative congruency of the objective functions to be minimized, it is also necessary to understand how feasible the objectives are w.r.t. the problem constraints. To this end, *objective feasibility* must be defined by approximating the constraint set in a meaningful way.

To develop this approximation, equality constraints must not be considered because they imply strict notions of what is feasible. Essentially, equality constraints do not admit any margin of feasibility/infeasibility as they are inherently binary in meaning. Therefore, if equality constraints exist, then they should be included in the resolution of each objective function — i.e. using (3.11) rather than (3.7), as explored in Chapter 4. This leaves the inequality constraints, $G\boldsymbol{x} \leq \boldsymbol{h}$, which, if consistent, form a convex polyhedron solution space. This space is a polytope bounded by the convex hull of the inequality halfspaces. The number of vertices of this polytope becomes combinatorially complex as the problem dimension increases, so it is necessary to search for a compact representation of the constraint set.

Feasibility Ellipsoid

Following the approach in Sec. 3.3.1, one solution is to find the largest ellipsoid which can be inscribed within the polyhedron defined by the inequality constraint set. This is a conservative approximation of the set and describes it with only $n \times n$ parameters, where n is the dimension of the optimization variable, $\boldsymbol{x}$.

Given a set of linear inequalities, $G\boldsymbol{x} \leq \boldsymbol{h}$, the objective is to find the maximum volume ellipsoid, parameterized by (3.52), which can be inscribed within polyhedron defined by the inequality half-spaces using,

$$\begin{aligned} \underset{B_\phi, \boldsymbol{c}_\phi}{\arg\min} \quad & \log \det B_\phi^{-1} \\ \text{s.t.} \quad & \left\| B_\phi \boldsymbol{g}_i \right\|_2 + \boldsymbol{g}_i^\top \boldsymbol{c}_\phi \leq h_i, \quad \forall i \in 1 \ldots n_c, \end{aligned} \tag{3.62}$$

where $\boldsymbol{g}_i$ and h_i are the i^{th} rows of G and $\boldsymbol{h}$, respectively, and n_c is the number of inequality constraints. Similarly to (3.51), the log det term is transformed into the n^{th} root determinant to make the problem solvable using SDP. See p. 414 of [Boyd and Vandenberghe, 2004] for more details. Applying (3.62) to the example problem, the feasibility ellipsoid depicted in Fig. 3.5a can be found. Again, the direction and length of semi-axes can be determined using the SVD of the B matrix.

The most feasible region of solutions is at the center of the feasibility ellipsoid, $\boldsymbol{c}_\phi$, while the least feasible regions are near its surface. Here the concept of feasibility is a continuous notion of how possible it is to perfectly optimize one or more objective functions. Firstly, in order to ensure *absolute objective feasibility*, all of the unconstrained objective optima, $\boldsymbol{x}_i^*$, must lie inside the feasibility ellipsoid. This can be verified by evaluating the feasibility ellipsoid inequality for each $\boldsymbol{x}_i^*$,

$$\begin{gathered} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi \right)^\top P_\phi^{-1} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi \right) \leq 1 \\ \text{where} \quad B_\phi^{-2} = P_\phi^{-1}. \end{gathered} \tag{3.63}$$

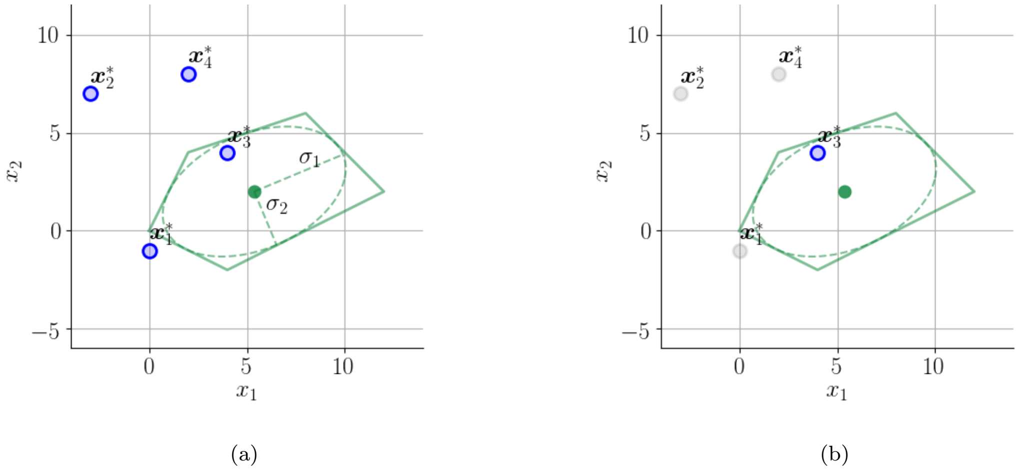


Figure 3.5 – (a) the feasibility ellipsoid for the example problem from Sec. 3.2.4. (b) evaluation of (3.63) for the example problem shows that only $\boldsymbol{x}_3^*$ lies within the feasibility ellipsoid and concordantly, inside the feasible set defined by the inequality constraints.

In the case of the example problem, shown in Fig. 3.5b, the only objective optimum which respects (3.63) is $\boldsymbol{x}_3^*$.

If (3.63) is true for a given $\boldsymbol{x}_i^*$ then it is certain that $f_i(\boldsymbol{x})$ can be perfectly minimized (assuming it is consistent) given the current problem constraints. However, it is not guaranteed that the unconstrained optimum of a combination of multiple objective functions, whose $\boldsymbol{x}_i^*$ respect (3.63), will fall entirely within the feasibility ellipsoid. If each E_i and its corresponding $\mathbf{f}_i$ satisfy the following condition,

$$\operatorname{rank}(E_i) = \operatorname{rank}([E_i|\mathbf{f}_i]) = n \ \forall \ i \in 1 \ldots n_o, \tag{3.64}$$

then each $f_i(\boldsymbol{x})$ admits only one solution for $\boldsymbol{x}_i^*$, and if those $\boldsymbol{x}_i^*$ respect (3.63), then they will be contained within the convex feasible set. As a reminder n_o is the number of objective functions. In turn, these $\boldsymbol{x}_i^*$ will form a convex set within the feasible set, and any solution to the multi-objective optimization will exist in this set. On the other hand, if one or more E_i and $\mathbf{f}_i$ produce the following relationship,

$$\operatorname{rank}(E_i) = \operatorname{rank}([E_i|\mathbf{f}_i]) < n, \tag{3.65}$$

then the $f_i(\boldsymbol{x})$ objective function admits infinitely many solutions and those solutions exist on a hyper-plane defined by the null-space of E_i. This hyper-plane is in no way guaranteed to be bounded by the feasible set, and could therefore produce an infeasible unconstrained optimum to the multi-objective problem. This is illustrated by Fig. 3.6, where the $\mathbf{f}_i$ values

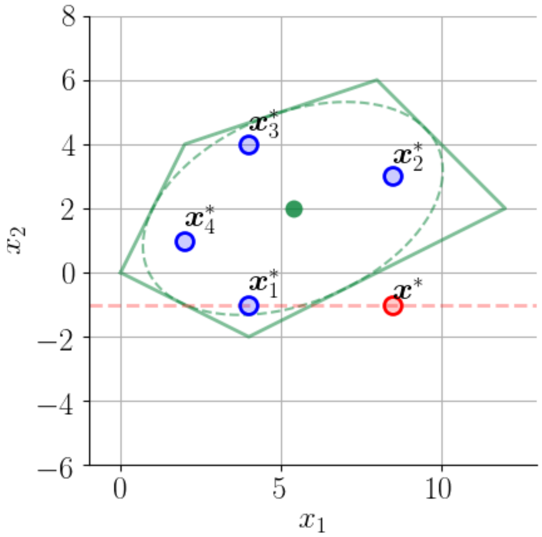


Figure 3.6 – An example showing that even if the unconstrained objective optima lie within the feasibility ellipsoid, then there is still the possibility that their combination will result in an infeasible optimal value. Here, hierarchical optimization is used to produce $\boldsymbol{x}^*$, but this same result could be found with the right combinations of weights as well.

of the example problem have been modified to the following,

$$\mathbf{f}_1 = \begin{bmatrix} 4 \\ -1 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} 8.5 \\ 3 \end{bmatrix} \quad \mathbf{f}_3 = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad \mathbf{f}_4 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}. \tag{3.66}$$

Keeping the E_i values the same and solving the multi-objective optimization problem hierarchically starting with $f_1(\boldsymbol{x})$ and ending with $f_4(\boldsymbol{x})$, produces an optimum, $\boldsymbol{x}^* = \begin{bmatrix} 8.5 & -1 \end{bmatrix}^\top$, which violates the inequality constraints despite all of the unconstrained objective optima being within the feasibility ellipsoid. This is the result of the null-space in the E_1 matrix, indicated by the dashed red line.

One obvious problem with the feasibility ellipsoid is that it does not closely approximate the space near the constraint polyhedron vertices, which can have large volumes when $n > 3$. Ideally, one would calculate the Löwner John ellipsoid for the convex polyhedron defined by the inequality constraints to obtain a less conservative approximation which encompasses the vertices; however, this problem is NP-hard. Fortunately, given the maximum volume inscribed ellipsoid from (3.5a), simply scaling it by a factor of n (the dimension of $\boldsymbol{x}$), assures that the entire polyhedron will be covered by the ellipsoid [Boyd and Vandenberghe, 2004]. In Fig 3.7, the scaled feasibility ellipsoid is presented for the example problem. As can be observed, the scaled ellipsoid indicates that both $\boldsymbol{x}_1^*$ and $\boldsymbol{x}_3^*$ should be feasible; however, it is clear in this example that although $\boldsymbol{x}_1^*$ is close to the true feasible set, it does not lie within it, so this scaled ellipsoid measure should be used with caution.

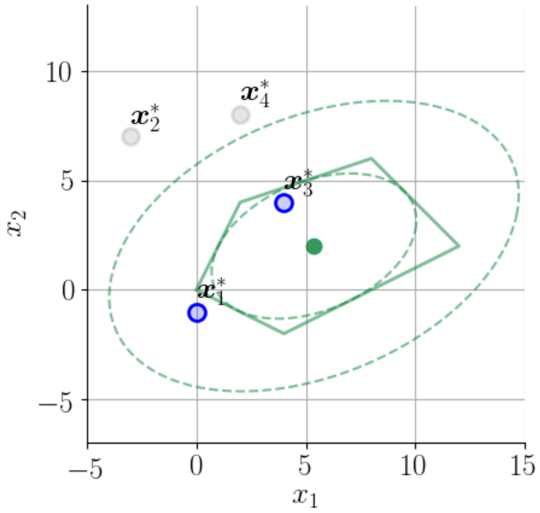


Figure 3.7 – Scaled feasibility ellipsoid using $n = 2$ since $\boldsymbol{x} \in \mathbb{R}^2$. Evaluation of (3.63) shows now that both $\boldsymbol{x}_1^*$ and $\boldsymbol{x}_3^*$ lie within the scaled feasibility ellipsoid; however, $\boldsymbol{x}_1^*$ is outside of the true feasible set defined by the inequality constraints.

Objective Feasibility Metrics

As with objective compatibility, it is also necessary to be able to measure the level of objective feasibility keeping in mind that feasibility is maximal at the center of the feasibility ellipsoid. Measures of objective feasibility, ϕ, are thus developed. An initial option is to calculate the *Optima Distance (OD)* between each $\boldsymbol{x}_i^*$ and the feasibility ellipsoid center,

$$\phi_i^{\mathrm{OD}} = \|\boldsymbol{c}_\phi - \boldsymbol{x}_i^*\| \;\; \forall i \in 1 \ldots n_o. \tag{3.67}$$

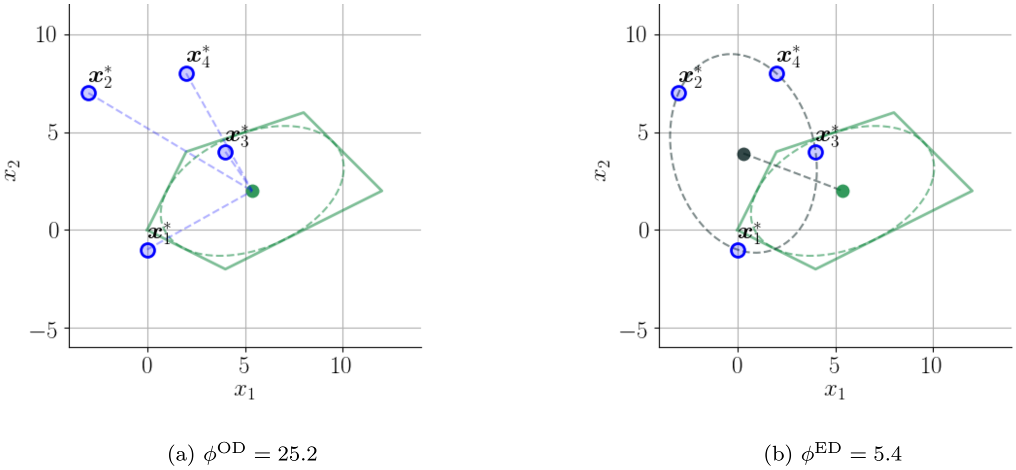
The sum of the ϕ_i^{OD} can then be used as a global objective feasibility metric,

$$\phi^{\mathrm{OD}} = \sum_{i=1}^{n_o} \phi_i. \tag{3.68}$$

For a more concise alternative, the *Ellipsoid Distance (ED)*, or the distance between the compatibility ellipsoid and the feasibility ellipsoid centers can be determined,

$$\phi^{\mathrm{ED}} = \|\boldsymbol{c}_\kappa - \boldsymbol{c}_\phi\|_2. \tag{3.69}$$

Figure 3.8 shows the evaluation of these two objective feasibility metrics for the example problem. Both ϕ^{OD} and ϕ^{ED} provide distance values which can be interpreted as feasibility values. Unfortunately, without a basis for comparison, there is no way of knowing if a single value indicated by the metric is feasible or not. Thus, a comparison of two or more metric values is necessary to determine a relative notion of which objectives are more feasible than others.



(a) $\phi^{\mathrm{OD}} = 25.2$ (b) $\phi^{\mathrm{ED}} = 5.4$

Figure 3.8 – Evaluation of the two proposed objective feasibility measures. (a) shows the metric defined by Equations (3.67) and (3.68), and (b) shows the metric defined by (3.69).

It would be useful to know whether a single optimum is feasible or not. To do so, the feasibility ellipsoid equation, (3.63), can be evaluated at each objective optimum,

$$\phi_i^{\mathrm{EE}} = \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right) \ \forall i \in 1 \dots n_o, \tag{3.70}$$

as well as at the overall unconstrained optimum,

$$\phi^{\mathrm{EE}^*} = \left(\boldsymbol{x}^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{x}^* - \boldsymbol{c}_\phi\right). \tag{3.71}$$

These metrics are denoted the *Ellipsoid Evaluation (EE)* metrics and can be seen as a weighted version of the ϕ^{OD} metrics where the ellipsoid matrix, P_ϕ^{-1}, is used to weight the squared norm of the distance. What is useful about this measure is that any value ≤ 1.0 indicates that the $\boldsymbol{x}_i^*$ is on or inside the feasibility ellipsoid, and any value > 1.0 means that $\boldsymbol{x}_i^*$ is outside of the feasibility ellipsoid. This allows each objective to be analyzed as feasible or infeasible according to the ellipse. This estimate of feasibility is conservative. To obtain a less conservative estimate, the ellipsoid can be scaled by n to obtain an enclosing ellipse,

$$\phi_i^{\mathrm{EE}} = \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right)^\top \left(nB_\phi\right)^{-2} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right) \ \forall i \in 1 \dots n_o. \tag{3.72}$$

Finally, to observe the evolution of the constraint set, the feasibility *Ellipsoid Volume (EV)* can be calculated to estimate how large the feasible set is, and how likely it is that an objective optimum will fall inside of it,

$$\phi^{\mathrm{EV}} = \det(B_\phi). \tag{3.73}$$

However, the measure ϕ^{EV} does not indicate how feasible a particular optimum is w.r.t. the feasibility ellipsoid, just how large or small the polyhedron defined by the constraints has become.

3.4 Conclusions

In this chapter we have looked at what makes tasks incompatible or infeasible from a pure optimization standpoint. By ignoring the control aspect of whole-body control and considering only its formulation as a multi-objective optimization problem, metrics are developed for measuring the compatibility and feasibility of a set of objectives and constraints. Tables A.3 and A.4 in Sec. A.2 of the Appendix summarize these metrics for the reader's convenience.

Ignoring the control aspects of the task compatibility and feasibility problem allows generic multi-objective optimization metrics to be constructed with no priors on the control formulation. What remains is to apply these metrics to the problem of redundant robot control and explore their ability to accurately describe the phenomena they are designed to measure.

Chapter 4

A Model-Based Approach to Maximizing Task Compatibility and Feasibility

In this chapter, the metrics developed to analyze the compatibility and feasibility of objective functions are applied to the problem of whole-body control. Using the whole-body controller objectives and constraint equations, the compatibility and feasibility metrics are reformulated, and implementation issues are discussed. By examining the underlying concepts behind the metrics, a whole-body Non-linear Model Predictive Control (NMPC) problem is developed which could theoretically maximize task compatibility and feasibility. Unfortunately, the computational complexity of the problem prohibits it from being an effective solution to real world problems.

4.1 Introduction

In Chapter 3, a number of compatibility and feasibility metrics are developed based purely on multi-objective optimization concepts. Compatibility indicates how possible it is for two or more objectives to be perfectly minimized. Feasibility indicates how well each objective can be minimized, given the problem constraints. The goal of these metrics is to provide information on how possible it is to accomplish all the desired objectives for a given optimization problem. In the context of whole-body control, this corresponds to a single control time step or one computation of the control problem,

$$\tilde{\boldsymbol{\chi}}^* = \arg\min_{\boldsymbol{\chi}} \quad \sum_{i=1}^{n_{\text{task}}} w_i f_i^{\text{task}}(\boldsymbol{\chi}) + w_0 f_0^{\text{task}}(\boldsymbol{\chi}) \tag{4.1}$$

$$\text{s.t.} \quad G\boldsymbol{\chi} \leq \boldsymbol{h} \tag{4.2}$$

$$A\boldsymbol{\chi} = \boldsymbol{b}. \tag{4.3}$$

In (4.1), a generic weighted sum prioritization whole-body controller is presented; however, the proceeding analyses remain valid for a hierarchical controller as well. As a reminder, $\boldsymbol{\chi}$ is the optimization variable which contains joint accelerations, torques and external wrenches, $f_i^{\text{task}}(\boldsymbol{\chi})$ are the task objective functions, w_i their weights or priorities, and $w_0 f_0^{\text{task}}(\boldsymbol{\chi})$ the regularization task. The inequality and equality constraints, (4.2) and (4.3), are composed of the constraints described in Sec. 2.3 such as joint position and torque constraints, and the dynamics. Here, we examine only the weighted prioritization scheme but the concepts developed apply to any convex combination or cascade of objectives. The use of the tilde notation in $\tilde{\boldsymbol{\chi}}^*$, to represent the solution to the problem, is explained in Sec 4.2.2.

The objective of this Chapter is to apply the metrics developed in Chapter 3 to the whole-body controller detailed in Chapter 2, and determine a method of maximizing task compatibility and feasibility. Because the concepts developed are based on the structure of the whole-body controller optimization problem, we refer to them as model-based. Beyond simply equating the formulae of the controller to the compatibility and feasibility metrics, it is necessary to develop a notion of the temporality of these measures given that an optimization problem is solved at each control time step and there are generally many time steps in a single movement.

4.2 Computing the Metrics

To begin, the discrepancies between the whole-body controller formulae in Chapter 2 and the metrics described in Chapter 3 must be reconciled. For convenience, Table A.1 and Table A.2 are provided to remind the reader of the relevant whole-body control formulae.

4.2.1 Equating Representations

It is chosen at the outset of this work to use the least-squares formulation of QPs. Thanks to this decision, the metrics described in Chapter 3 have an almost one to one variable correspondence with the equations presented in Chapter 2. The most notable exception to this is the optimization variable which is $\boldsymbol{x}$ in Chapter 3 and $\boldsymbol{\chi}$ in Chapter 2. Through a simple redefinition of the optimization variable, i.e. $\boldsymbol{x} := \boldsymbol{\chi}$, the compatibility and feasibility metrics can be rewritten as shown in Tables 4.1 and 4.2. All variables maintain the same definitions as explained in Chapter 3; however, the unconstrained objective (task) optima, $\boldsymbol{\chi}_i^*$, require a bit more treatment due to the existence of the equations of motion equality constraints.

Compatibility Metric	**Mathematical Formulae**
Nuclear norm Ratio (NR)	$\kappa^{\text{NR}} = \frac{\left\Vert \tilde{E} \right\Vert_*}{\left\Vert [\tilde{E} \vert \tilde{\mathbf{f}}] \right\Vert_*}$
Optimum Distance (OD)	$\kappa^{\text{OD}} = \sum_{i=1}^{n_o} \kappa_i^{\text{OD}} = \sum_{i=1}^{n_o} \Vert \boldsymbol{\chi}^* - \boldsymbol{\chi}_i^* \Vert_2$
Optimum Cost (OC)	$\kappa^{\text{OC}} = \sum_{i=1}^{n_o} \kappa_i^{\text{OC}} = \sum_{i=1}^{n_o} w_i f_i(\boldsymbol{\chi}^*)$
Ellipsoid Volume (EV)	$\kappa^{\text{EV}} = \det(B_\kappa) = \prod_{i=1}^{n} \sigma_i$
Ellipsoid Distance (ED)	$\kappa^{\text{ED}} = \sum_{i=1}^{n_o} \kappa_i^{\text{ED}} = \sum_{i=1}^{n_o} \Vert \boldsymbol{c}_\kappa - \boldsymbol{\chi}_i^* \Vert_2$

Table 4.1 – Compatibility metrics formulated using the whole-body controller variables from Chapter 2.

Feasibility Metric	**Mathematical Formulae**
Optima Distance (OD)	$\phi^{\text{OD}} = \sum_{i=1}^{n_o} \phi_i^{\text{OD}} = \sum_{i=1}^{n_o} \lVert \boldsymbol{c}_\phi - \boldsymbol{\chi}_i^* \rVert_2$
Ellipsoid Distance (ED)	$\phi^{\text{ED}} = \lVert \boldsymbol{c}_\kappa - \boldsymbol{c}_\phi \rVert_2$
Ellipsoid Evaluation (EE)	$\phi^{\text{EE}} = \sum_{i=1}^{n_o} \phi_i^{\text{EE}} = \sum_{i=1}^{n_o} \left(\boldsymbol{\chi}_i^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{\chi}_i^* - \boldsymbol{c}_\phi\right)$
Ellipsoid Evaluation (EE)	$\phi^{\text{EE}^*} = \left(\boldsymbol{\chi}^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{\chi}^* - \boldsymbol{c}_\phi\right)$
Ellipsoid Volume (EV)	$\phi^{\text{EV}} = \det(B_\phi) = \prod_{i=1}^{n_c} \sigma_i$

Table 4.2 – Feasibility metrics formulated using the whole-body controller variables from Chapter 2.

4.2.2 Handling Equality Constraints

In the whole-body controller, equality constraints on $\boldsymbol{\chi}$ are used to respect the equations of motion and no relative motion at contact locations. As a reminder the equality constraint for the equation of motion from (2.33) is expressed as,

$$\underbrace{\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^{e}J^\top(\boldsymbol{q}) \end{bmatrix}}_{A^d} \boldsymbol{\chi} = \underbrace{\boldsymbol{n}(\boldsymbol{q}, \boldsymbol{\nu})}_{\boldsymbol{b}^d},$$

and the contact constraint from (2.47) which ensures no relative motion of the contact point is,

$$\underbrace{\begin{bmatrix} {}^{e}J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix}}_{A^\omega} \boldsymbol{\chi} = \underbrace{-{}^{e}\dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu}}_{\boldsymbol{b}^\omega}.$$

While it is possible to project these constraints into the objective functions, difficulties can arise in the free-floating case (see [Sentis and Khatib, 2005; Mistry et al., 2010]). Therefore, the equality constraints are used explicitly here. Because of these equality constraints, the unconstrained objective function optima, which are the task optima, cannot be calculated as $\boldsymbol{\chi}_i^* = E_i^\dagger \mathbf{f}_i$ because the equality constraints are not taken into consideration. Luckily, the equality constrained least-squares problem admits an analytical solution (when one exists) as shown in (3.11), which can be applied to the whole-body controller task optima,

$$\underbrace{\begin{bmatrix} E^\top E & A^\top \\ A & \mathbf{0} \end{bmatrix}}_{\hat{E}} \underbrace{\begin{bmatrix} \boldsymbol{\chi} \\ \boldsymbol{z} \end{bmatrix}}_{\hat{\boldsymbol{\chi}}} = \underbrace{\begin{bmatrix} E^\top \mathbf{f} \\ \boldsymbol{b} \end{bmatrix}}_{\hat{\mathbf{f}}} \tag{4.4}$$

$$\Leftrightarrow \hat{\boldsymbol{\chi}}^* = \hat{E}^{-1}\hat{\mathbf{f}}. \tag{4.5}$$

In (4.4), $\boldsymbol{z}$ are the Lagrange multipliers and a solution exists if $\hat{E}$ is invertible. It should be noted that redundancy in the DoF must be dealt with through regularization of the least-squares problem,

$$f(\boldsymbol{\chi}) = \|E\boldsymbol{\chi} - \mathbf{f}\| + w_0 \|\boldsymbol{\chi}\|, \tag{4.6}$$

where $w_0 \ll 1$ is the regularization weight. The regularization objective function can be incorporated into (4.10) by redefining E and $\mathbf{f}$ as,

$$E := \begin{bmatrix} E \\ \sqrt{w_0} I \end{bmatrix} \tag{4.7}$$

$$\mathbf{f} := \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}, \tag{4.8}$$

with $I \in \mathbb{R}^{n\times n}$, where n is the dimension of $\boldsymbol{\chi}$, and $\mathbf{0} \in \mathbb{R}^n$. Note that the regularization objective can be something other than $\mathbf{0}$. Using (4.5) for an individual task provides the equality constrained optimum, $\hat{\boldsymbol{\chi}}_i^*$ for that task,

$$\hat{\boldsymbol{\chi}}_i^* = \hat{E}_i^{-1}\hat{\mathbf{f}}_i, \tag{4.9}$$

where the first n rows of $\hat{\boldsymbol{\chi}}_i^*$, are the primal objective optimum, $\boldsymbol{\chi}_i^*$, which can be obtained with,

$$\boldsymbol{\chi}_i^* = S^n\hat{\boldsymbol{\chi}}_i^* = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \hat{E}_i^{-1}\hat{\mathbf{f}}_i. \tag{4.10}$$

In (4.10), S^n is a selection matrix used to extract the first n rows of $\hat{\boldsymbol{\chi}}_i^*$, and $S^n = \begin{bmatrix} I & \mathbf{0} \end{bmatrix}$ with $I \in \mathbb{R}^{n\times n}$. Because the unconstrained objective optima is no longer used in the analysis of compatibility and feasibility, it is necessary to properly distinguish between the optima found from the analytical resolution of each task objective function and the numerical solution to the constrained whole-body control problem from (4.1). To do so, the following definitions are created,

Task Optimum: ($\boldsymbol{\chi}_i^*$) The analytical equality constrained optimum of a single task objective function, determined using (4.10).

Prioritized Optimum: ($\boldsymbol{\chi}^*$) The analytical equality constrained optimum of the prioritized combination of task objective functions, determined by solving the problem in (4.1), without the inequality constraints.

Controller Optimum: ($\tilde{\boldsymbol{\chi}}^*$) The numerical equality and inequality constrained optimum of a combination of task objective functions, determined by solving (4.1).

These definitions should help clarify the following sections.

Finally, to calculate the Nuclear norm Ratio, κ^{NR}, $\tilde{E}$ and $\tilde{\mathbf{f}}$ must be formed. For this, two options are available. The first is simply to use the equations described in 3.26 directly, and form the two variables with only the task E_i and $\mathbf{f}_i$ $\forall i \in 1 \ldots n_{\text{task}}$, i.e.

$$\tilde{E} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n_{\text{task}}} \end{bmatrix} \quad \tilde{\mathbf{f}} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_{\text{task}}} \end{bmatrix}. \tag{4.11}$$

This should indicate how compatible the tasks are without regards to the equations of motion. The second option then, is to include the equation of motion and contact equality constraints from (2.33) and (2.47),

$$\tilde{E}_A = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n_{\text{task}}} \\ A_d \\ A_{\boldsymbol{\omega}} \end{bmatrix} \quad \tilde{\mathbf{f}}_A = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_{\text{task}}} \\ \boldsymbol{b}_d \\ \boldsymbol{b}_{\boldsymbol{\omega}} \end{bmatrix}, \tag{4.12}$$

to form the augmented versions of $\tilde{E}$ and $\tilde{\mathbf{f}}$ and solve the *Augmented Nuclear norm Ratio*,

$$\kappa_A^{\mathrm{NR}} = \frac{\left\| \tilde{E}_A \right\|_*}{\left\| \begin{bmatrix} \tilde{E}_A | \tilde{\mathbf{f}}_A \end{bmatrix} \right\|_*}. \tag{4.13}$$

Equation (4.13) should indicate how compatible the tasks are with one another, the system dynamics, and any contact constraints.

4.2.3 Computing the Compatibility and Feasibility Ellipsoids

Calculating the compatibility and feasibility ellipsoids for the whole-body control problem is essentially the same as the examples given in Chapter 3. The generic definition of the $\boldsymbol{\chi}_i^*$ from (4.10) can be applied to each task type,

$$\boldsymbol{\chi}^{\ddot{\boldsymbol{\xi}}^*} = S^n \hat{E}^{\ddot{\boldsymbol{\xi}}^{-1}} \hat{\mathbf{f}}^{\ddot{\boldsymbol{\xi}}}, \tag{4.14}$$

$$\boldsymbol{\chi}^{\boldsymbol{\omega}^*} = S^n \hat{E}^{\boldsymbol{\omega}^{-1}} \hat{\mathbf{f}}^{\boldsymbol{\omega}}, \tag{4.15}$$

$$\text{etc.,}$$

because each has been formulated in the same least-squares manner. Dropping the task type superscripts, the compatibility ellipsoid is computed by combining all of the n_{task} task optima into the set $\mathcal{C}$,

$$\mathcal{C} = \left\{\boldsymbol{\chi}_1^*, \boldsymbol{\chi}_2^*, \ldots, \boldsymbol{\chi}_{n_{\text{task}}}^*\right\} \tag{4.16}$$

$$= \left\{S^n \hat{E}_1^{-1} \hat{\mathbf{f}}_1, S^n \hat{E}_2^{-1} \hat{\mathbf{f}}_2, \ldots, S^n \hat{E}_{n_{\text{task}}}^{-1} \hat{\mathbf{f}}_{n_{\text{task}}}\right\}. \tag{4.17}$$

and solving the SDP detailed in (3.51).

Computing the feasibility ellipsoid is similarly straightforward and requires only the concatenation of the relevant inequality constraints. These constraints come from those described in Sec. 2.3 and only those which are actively being used on the robot need to figure into the computation of the feasibility ellipsoid. Thus, given the final $G\boldsymbol{\chi} \leq \boldsymbol{h}$ used in (4.1), the feasibility ellipsoid can be computed through direct application of (3.62).

4.3 The Temporality of Compatibility and Feasibility

The compatibility and feasibility metrics and measures developed in this study have, until now, been regarded from the optic of a single optimization problem. In whole-body control, this represents one control time step and therefore, a very small part of a whole-body motion. A single control time step is on the order of 1ms, and can be seen as an instant in time w.r.t. timescales commensurate with robotic applications. Given that a single optimization problem represents an instant in time, the metrics designed here to understand the task compatibility and feasibility of the problem are only valid at that instant. By computing the metrics over the duration of a movement, then a timeline of compatibility and feasibility is constructed. This timeline should provide an idea of how the task compatibility and feasibility evolves with the control problem. This introduces the notion of temporality to the concepts of compatibility and feasibility — something unique to control applications. As the tasks track their references which come from the servoing of task trajectories, their compatibility and feasibility will evolve and change. Thus, a clear link between task trajectories and task compatibility and feasibility is created, allowing us to formulate a method for maximizing the task compatibility and feasibility.

4.4 Maximizing Task Compatibility and Feasibility

The goal of this work is to provide a mechanism by which task compatibility and feasibility can be maximized for a given whole-body control architecture, by observing the performance of the whole-body motion and optimizing the task trajectories to maximize task compatibility and feasibility. Given the various task compatibility and feasibility metrics developed in this chapter, a problem could be formulated using any combination of these metrics as optimality criteria. However, two simple notions can be extracted from the developed metrics. Firstly, task compatibility is at a maximum when the task optima, $\boldsymbol{\chi}_i^*$ are all coincident, or in the null spaces of the other task objective functions. Secondly, task feasibility is maximized when the tasks are near the feasibility ellipsoid center, i.e. far from the control constraints. We can easily combine these two concepts by trying to move the task optima, $\boldsymbol{\chi}_i^*$, to the center of the feasibility ellipsoid, $\boldsymbol{c}_\phi$. Doing so, would shrink the space between the task optima, thereby improving their compatibility, and distance the task optima from the constraints, thus improving their feasibility. Therefore, to maximize task compatibility and feasibility, the following optimization problem is formulated,

$$\underset{\check{\boldsymbol{\chi}}_i(k)}{\arg\min} \quad \|\check{\boldsymbol{\chi}}_i(k) - \boldsymbol{c}_\phi(k)\|_2^2 \tag{4.18}$$

$$\text{s.t.} \quad A(k)^d \check{\boldsymbol{\chi}}_i(k) = \boldsymbol{b}^d(k), \tag{4.19}$$

where k is the control time step, and $\check{\boldsymbol{\chi}}_i$ is the objective variable. The equality constraint in (4.19) contains the equations of motion for the system from (2.9), which are repeated here:

$$\underbrace{\begin{bmatrix} -M(\boldsymbol{q}(k)) & S^\top & {}^eJ^\top(\boldsymbol{q}(k)) \end{bmatrix}}_{A^d} \check{\boldsymbol{\chi}}_i(k) = \underbrace{\boldsymbol{n}(\boldsymbol{q}(k), \boldsymbol{\nu}(k))}_{\boldsymbol{b}^d}. \tag{4.20}$$

This optimization can be performed at each time step over the full control horizon of the movement, $k = 0, 1, \ldots, n_h$,

$$\mathbb{X}_i^* = \underset{\mathbb{X}_i}{\arg\min} \quad \|\mathbb{X}_i - \mathbb{C}\|_2^2 \tag{4.21}$$

$$\text{s.t.} \quad \mathbb{A}\mathbb{X}_i = \mathbb{B} \tag{4.22}$$

where,

$$\mathbb{X}_i = \begin{bmatrix} \check{\boldsymbol{\chi}}_i(k) \\ \check{\boldsymbol{\chi}}_i(k+1) \\ \vdots \\ \check{\boldsymbol{\chi}}_i(k+n_h-1) \end{bmatrix} \quad \mathbb{C} = \begin{bmatrix} \boldsymbol{c}_\phi(k) \\ \boldsymbol{c}_\phi(k+1) \\ \vdots \\ \boldsymbol{c}_\phi(k+n_h-1) \end{bmatrix} \quad \mathbb{A} = \begin{bmatrix} A(k) \\ A(k+1) \\ \vdots \\ A(k+n_h-1) \end{bmatrix} \quad \mathbb{B} = \begin{bmatrix} \boldsymbol{b}(k) \\ \boldsymbol{b}(k+1) \\ \vdots \\ \boldsymbol{b}(k+n_h-1) \end{bmatrix}. \tag{4.23}$$

Equation (4.21) defines a whole-body NMPC problem. The optimum of this problem, $\mathbb{X}_i^*$, then contains all of the new objective optima which would maximize the i^{th} task's compatibility and feasibility,

$$\mathbb{X}_i^* = \begin{bmatrix} \check{\boldsymbol{\chi}}_i^*(k) \\ \check{\boldsymbol{\chi}}_i^*(k+1) \\ \vdots \\ \check{\boldsymbol{\chi}}_i^*(k+n_h-1) \end{bmatrix}. \tag{4.24}$$

By optimizing $\mathbb{X}_i$ in (4.21), the task trajectory is modified. While this is the ultimate goal of our work, the tasks must nevertheless attain their specified goals. These goals are typically specified by the trajectory waypoints. If (4.21) is solved, then it is likely that the optimized task trajectory will deviate wildly from the original task trajectory. To avoid this behavior, and ensure that all task waypoints are achieved, constraints must be placed on $\mathbb{X}_i$. Here, constraints are developed for an operational-space acceleration task, but constraints can also be applied to joint-space acceleration tasks and wrench tasks using similar techniques. To enforce position constraints on an operational-space acceleration task, the relationship from (2.15) can be rewritten,

$$E_i^{\ddot{\boldsymbol{\xi}}}(k)\check{\boldsymbol{\chi}}_i(k) = \mathbf{f}_i^{\ddot{\boldsymbol{\xi}}}(k), \tag{4.25}$$

with

$$E_i^{\ddot{\boldsymbol{\xi}}}(k) = \begin{bmatrix} J_i(\boldsymbol{q}(k)) & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{f}_i^{\ddot{\boldsymbol{\xi}}}(k) = \ddot{\boldsymbol{\xi}}_i^{\text{des}}(k) - \dot{J}_i(\boldsymbol{q}(k), \boldsymbol{\nu}(k))\boldsymbol{\nu}(k). \tag{4.26}$$

Rearranging terms gives,

$$\ddot{\boldsymbol{\xi}}_i^{\text{des}}(k) = E_i^{\ddot{\boldsymbol{\xi}}}(k)\check{\boldsymbol{\chi}}_i(k) + \dot{J}_i(\boldsymbol{q}(k), \boldsymbol{\nu}(k))\boldsymbol{\nu}(k). \tag{4.27}$$

Given position constraints on the task trajectory in the form of,

$$\boldsymbol{\xi}_{i_{\min}}(k) \leq \boldsymbol{\xi}^{\mathrm{des}}(k) \leq \boldsymbol{\xi}_{i_{\max}}(k), \tag{4.28}$$

and using discrete integration,

$$\boldsymbol{\xi}_i^{\mathrm{des}}(k) = \boldsymbol{\xi}_i^{\mathrm{des}}(k-1) + h\dot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k-1) + \frac{h^2}{2}\ddot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k-1) \tag{4.29}$$

$$\boldsymbol{\xi}_i^{\mathrm{des}}(k) = \boldsymbol{\xi}_i^{\mathrm{des}}(k-1) + h\dot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k-1) + \frac{h^2}{2}\Big(E_i^{\ddot{\boldsymbol{\xi}}}(k-1)\check{\boldsymbol{\chi}}_i(k-1) + \dot{J}_i(\boldsymbol{q}(k-1), \boldsymbol{\nu}(k-1))\boldsymbol{\nu}(k-1)\Big) \tag{4.30}$$

a relationship is established between task position constraints along the task trajectory and the optimization variable $\check{\boldsymbol{\chi}}_i$. Reformulating (4.28) and (4.30) produces the following inequality constraint,

$$\boldsymbol{\xi}_{i_{\min}}(k+1) \leq \boldsymbol{\xi}_i^{\mathrm{des}}(k) + h\dot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k) + \frac{h^2}{2}\Big(E_i^{\ddot{\boldsymbol{\xi}}}(k)\check{\boldsymbol{\chi}}_i(k) + \dot{J}_i(\boldsymbol{q}(k), \boldsymbol{\nu}(k))\boldsymbol{\nu}(k)\Big) \leq \boldsymbol{\xi}_{i_{\max}}(k+1) \tag{4.31}$$

$$\boldsymbol{h}_{\min}(k+1) \leq E_i^{\ddot{\boldsymbol{\xi}}}(k)\check{\boldsymbol{\chi}}_i(k) \leq \boldsymbol{h}_{\min}(k+1) \tag{4.32}$$

with

$$\boldsymbol{h}_{\min}(k+1) = \frac{2}{h^2}\Big(\boldsymbol{\xi}_{i_{\min}}(k+1) - \Big(\boldsymbol{\xi}_i^{\mathrm{des}}(k) + h\dot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k)\Big)\Big) - \dot{J}_i(\boldsymbol{q}(k), \boldsymbol{\nu}(k))\boldsymbol{\nu}(k) \tag{4.33}$$

$$\boldsymbol{h}_{\max}(k+1) = \frac{2}{h^2}\Big(\boldsymbol{\xi}_{i_{\max}}(k+1) - \Big(\boldsymbol{\xi}_i^{\mathrm{des}}(k) + h\dot{\boldsymbol{\xi}}_i^{\mathrm{des}}(k)\Big)\Big) - \dot{J}_i(\boldsymbol{q}(k), \boldsymbol{\nu}(k))\boldsymbol{\nu}(k). \tag{4.34}$$

Putting (4.32) in matrix form gives,

$$\underbrace{\begin{bmatrix} E_i^{\ddot{\boldsymbol{\xi}}}(k) \\ -E_i^{\ddot{\boldsymbol{\xi}}}(k) \end{bmatrix}}_{G_i} \check{\boldsymbol{\chi}}_i(k) \leq \underbrace{\begin{bmatrix} \boldsymbol{h}_{\max}(k+1) \\ -\boldsymbol{h}_{\min}(k+1) \end{bmatrix}}_{\boldsymbol{h}_i} \tag{4.35}$$

$$G_i(k)\check{\boldsymbol{\chi}}_i(k) \leq \boldsymbol{h}_i(k). \tag{4.36}$$

Finally, concatenating (4.36) over the entire control horizon using the variables,

$$\mathbb{G}_i = \begin{bmatrix} G_i(k) \\ G_i(k+1) \\ \vdots \\ G_i(k+n_h-1) \end{bmatrix} \quad \mathbb{H}_i = \begin{bmatrix} \boldsymbol{h}_i(k) \\ \boldsymbol{h}_i(k+1) \\ \vdots \\ \boldsymbol{h}_i(k+n_h-1) \end{bmatrix}, \tag{4.37}$$

produces the inequality constraint,

$$\mathbb{G}_i\mathbb{X}_i \leq \mathbb{H}_i. \tag{4.38}$$

Introducing (4.38) into (4.21), allows the NMPC problem to take into account position constraints on the task trajectory,

$$\mathbb{X}_i^* = \arg\min_{\mathbb{X}_i} \quad \|\mathbb{X}_i - \mathbb{C}\|_2^2 \quad \forall i \in 1 \ldots n_{\mathrm{task}} \tag{4.39}$$

$$\text{s.t.} \quad \mathbb{A}\mathbb{X}_i = \mathbb{B} \tag{4.40}$$

$$\mathbb{G}_i\mathbb{X}_i \leq \mathbb{H}_i. \tag{4.41}$$

The constraints defined by $\mathbb{G}_i\mathbb{X}_i = \mathbb{H}_i$ require that position constraints for each time step in the trajectory exist. At the waypoints along the trajectory the margin for error should be low and thus tightly constrained. Between these waypoints, there is less need for staying close to the original trajectory. During these phases of the trajectory, less stringent spatial bounds can be applied to the trajectory optimization. Methods for determining these bounds are not proposed here.

It should be noted that the optimization problem developed here is non-linear, non-convex, time-varying and possibly discontinuous. These factors make its resolution numerically complex, and potentially intractable in time scales which would be useful for robotics.

4.5 Conclusions

In this chapter, the objective compatibility and feasibility metrics developed in Chapter 3 are translated into task compatibility and feasibility metric through a few basic reformulations of the original metric definitions. However, these metrics remain to be tested on a real example and this is the goal of Chapter 6. It should be reiterated that the reason for developing these metrics is simply to better understand why tasks are incompatible and/or infeasible, and how this evolves over the course of the motion execution. Such insight would be invaluable for operators for tuning gains, parameters, and replanning.

Beyond simply understanding task compatibility and feasibility, the metrics also help us to understand how best to maximize task compatibility and feasibility. By developing the optimization problem in (4.39) which tries to move the task optima towards the feasibility ellipsoid center, task compatibility and feasibility could be maximized while ensuring that task trajectory waypoints are attained. This means that it is possible to seek task trajectories which improve both the compatibility and feasibility of the overall movement, while respecting the original intent of the movement. This is of course predicated on the assumption that one or more task trajectories can be modified, and that there exist such trajectories which improve task compatibility and feasibility.

Unfortunately, the full horizon whole-body NMPC problem defined by (4.39) is computationally difficult to solve due to its dimension, non-convexity, and time-dependence. While it is likely possible to determine a locally optimal solution to the problem, doing so would take too much time (as of this writing) to be used online. However, it could be used to optimize planned task trajectories for maximal compatibility and feasibility, prior to execution. A common alternative, is to solve (4.39) over a short preview horizon, rather than the whole trajectory. Reducing the problem dimension might make it tractable in task servoing time scales, given the many advances in whole-body NMPC (see Sec. 1.3.3).

Besides the complexity of any whole-body NMPC problem, solving (4.39) is particularly challenging because it requires the computation of the feasibility ellipsoid at each time step. While the feasibility ellipsoid is useful for compactly describing the constraint set, it may be overkill when only the approximate center is needed. To this end, another approach would be to use the analytic center, which can be found using,

$$\underset{\boldsymbol{\chi}}{\arg\min} \quad -\sum \left(\log\left(\boldsymbol{h}_{\mathrm{wbc}} - G_{\mathrm{wbc}}\boldsymbol{\chi}\right)\right), \tag{4.42}$$

where $\boldsymbol{h}_{\mathrm{wbc}}$ and G_{wbc} are the inequality constraint equations of the whole-body controller (see p. 434 of [Boyd and Vandenberghe, 2004]). The NMPC objective function from (4.18) could then be expressed as,

$$\underset{\check{\boldsymbol{\chi}}_i(k)}{\arg\min} \quad -\sum \left(\log\left(\boldsymbol{h}_{\mathrm{wbc}}(k) - G_{\mathrm{wbc}}(k)\check{\boldsymbol{\chi}}_i(k)\right)\right) \tag{4.43}$$

$$\text{s.t.} \quad A(k)^d\check{\boldsymbol{\chi}}_i(k) = \boldsymbol{b}^d(k) \tag{4.44}$$

$$G_i(k)\check{\boldsymbol{\chi}}_i(k) \leq \boldsymbol{h}_i(k), \tag{4.45}$$

which would drive the task optima towards the analytic center of the constraint set, while respecting the position bounds of the original task trajectory defined by (4.36). This computation is far less complex than that needed for the feasibility ellipsoid, and could make the proposed NMPC problem tractable. However, the complexities of integrating the whole-body dynamics at each optimization iteration remain an issue.

Sadly, even if this problem can be rendered numerically efficient, it still does not solve the issue of modeling errors and uncertainties. Despite the robustness of predictive methods, there will always be errors in the model used. These errors lead to incompatible and infeasible tasks and are only seen when the tasks are executed. For these reasons, a model-based solution to task compatibility and feasibility may not be possible or even desirable. Consequently, it is necessary to explore other avenues of resolution using model-free approaches.

Chapter 5

A Model-Free Approach to Maximizing Task Compatibility and Feasibility

---

Knowing that a model-based method for maximizing task compatibility and feasibility is numerically intractable for the moment, in this chapter, we approach the same problem from a model-free perspective. Specifically, the task servoing, whole-body control and low-level torque control are viewed as generic policies, which are parameterized by the task trajectories provided by the high-level task planning. From there, a simple series of cost functions are developed, which allow the side-effects (performance degradation) of task incompatibility and infeasibility to be measured. With the parameterized policies and cost functions, we develop a task compatibility and feasibility maximization algorithm, which uses model-free policy search techniques to optimize the task trajectories.

5.1 Introduction

The model-based techniques for measuring and optimizing task compatibility and feasibility developed in Chapter 4, provide insight as to why tasks are or are not executed as planned, and offer the potential to optimize them if needed. However, the complexity of the problem developed in Sec. 4.4 leads to a problem which may or may not be solvable in reasonable timescales. Furthermore, these model-based methods are subject to modeling errors. Consequently, model-based optimization may not be the best solution for removing task incompatibility and infeasibility, which is due in part to modeling errors [Prete and Mansard, 2016]. Because of these issues, the obvious solution is to use model-free techniques. More specifically, Model-Free Policy Search (MFPS), has been shown to be a powerful alternative to model-based planning (optimization) for dynamic robot tasks [Deisenroth et al., 2013].

In this chapter we show how MFPS can be integrated into the model-based control architecture presented in Fig. 1.3 in order to iteratively improve task trajectories to maximize task compatibility and feasibility. To do so, the task compatibility and feasibility problem is posed using the MFPS state-action formulation. The problem is formulated in such a way that any generic whole-body control architecture can be included into the control policy and by rolling out the policies, the real-world execution errors can be captured. The parameters of policy are specified at the task level and do not require any special task formulation. A series of cost functions are then developed to measure the effects of task incompatibility and infeasibility on the overall whole-body motion execution. Using the model-free formulation and cost functions, a *Task Compatibility and Feasibility Maximization* (TCFM) algorithm is developed. Important practical considerations, such as how to parameterize the policy, and different update strategies are explored. The formulation and TCFM algorithm are then demonstrated using a simple example problem.

5.2 Model-Free Formulation

To develop an MFPS algorithm for maximizing the compatibility and feasibility of one or more tasks, the problem must first be formulated in the standard MFPS fashion by defining the agent, environment, states, and actions. This helps define clearly how control policies can be developed, which exploit an underlying model-based control architecture. The components of MFPS are represented graphically in Fig. 5.1.

5.2.1 Agent & Environment

The *agent* is an abstract entity which decides what action, $\boldsymbol{a}(k)$, to take at time k, given the current state, $\boldsymbol{s}(k)$, of its *environment*. In this case, the environment consists of the robot and its surroundings, which are everything outside of the robot which can interact with or have an impact on the robot. The agent makes its decisions using a control *policy*, $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{s}(k))$. Here, the policy is the robot control architecture being used. More details on policies are provided in Sec. 5.2.3

5.2.2 States & Actions

The robot *state*, $\boldsymbol{s}(k)$ is captured by its $\boldsymbol{q}$ and $\boldsymbol{\nu}$ variables. These two pieces of information indicate the precise state of the robot w.r.t. some world frame, at the discrete control time step k. For the purposes of this study, the robot's surroundings are encoded by the wrenches and contact constraints used within the control. It may be advantageous for other applications to develop a more detailed encoding for the environment (e.g. depth map, symbolics, etc.) but here, the concern is how well tasks are performed without regard to why they are being performed, thus such encodings would only add superfluous data.

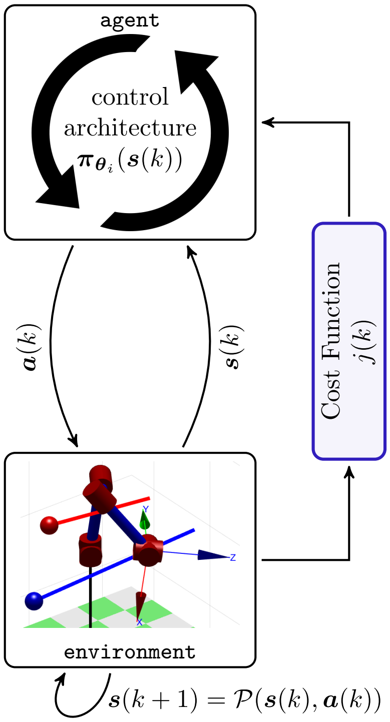

agent
control
architecture
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
$\boldsymbol{a}(k)$
$\boldsymbol{s}(k)$
Cost Function
$j(k)$
environment
$\boldsymbol{s}(k+1) = \mathcal{P}(\boldsymbol{s}(k), \boldsymbol{a}(k))$

Figure 5.1 – In MFPS, the problem is composed of an agent (control architecture) which interacts with its environment (robot + surroundings) using actions, $\boldsymbol{a}(k)$, the environment's state, $\boldsymbol{s}(k+1)$, changes and a cost, $j(k)$ is associated with that interaction.

The *transition dynamics*, $\boldsymbol{s}(k+1) = \mathcal{P}(\boldsymbol{s}(k), \boldsymbol{a}(k))$, are governed by continuous time second order dynamics; however, because the states must be sampled in order to control the robot (*discrete time control*) they are represented as discrete quantities. The application and removal of contacts, whether internal to the robot or between the robot and environment, produces discontinuities in these dynamics. The *actions*, $\boldsymbol{a}(k)$, are the actuator torques, developed at each control instant, $\boldsymbol{a}(k) = \boldsymbol{\tau}(k)$. The actuator torques cause the robot to move, changing its state, and possibly changing its state of interaction with the environment (i.e. contacts). Torques are considered here, but forces (linear actuation) can also be part of the agent's action vector. Again, these actions are generated using discrete time control, and are therefore discrete in the formulation.

5.2.3 Policies

We assume that a model-based whole-body control architecture is being used to control the robot. The policies,

$$\boldsymbol{a}(k) = \boldsymbol{\pi}\left(\boldsymbol{s}(k)\right), \tag{5.1}$$

determine the action at time k given the current state of the robot in its environment[1]. Since the actions are the torques computed by the whole-body controller and the states the robot's state and its contacts with the environment, then the policy must contain the whole-body controller. The whole-body controller is provided with desired task values by the task servoing level, therefore any task servoing controllers being used are also contained within $\boldsymbol{\pi}\left(\boldsymbol{s}(k)\right)$. Moving up the control hierarchy, the task servoing level is fed with reference task values by the task trajectories. The policies, therefore, provide mappings from task reference inputs, $\dot{\boldsymbol{\nu}}_i^{\text{ref}}, \ddot{\boldsymbol{\xi}}_i^{\text{ref}}$, etc. $\forall i \in 1, 2, \ldots, n_{\text{task}}$, to actuator torques, $\boldsymbol{\tau}$. This mapping is detailed in Fig. 5.2, in which all of the internal workings of, $\boldsymbol{\pi}\left(\boldsymbol{s}(k)\right)$, are indicated by the red bounding box.

Formulating the control policies in this manner means that they are directly dictated by the task trajectories. Thus, whatever parameterization is used to generate the task trajectories is therefore the policy parameterization.

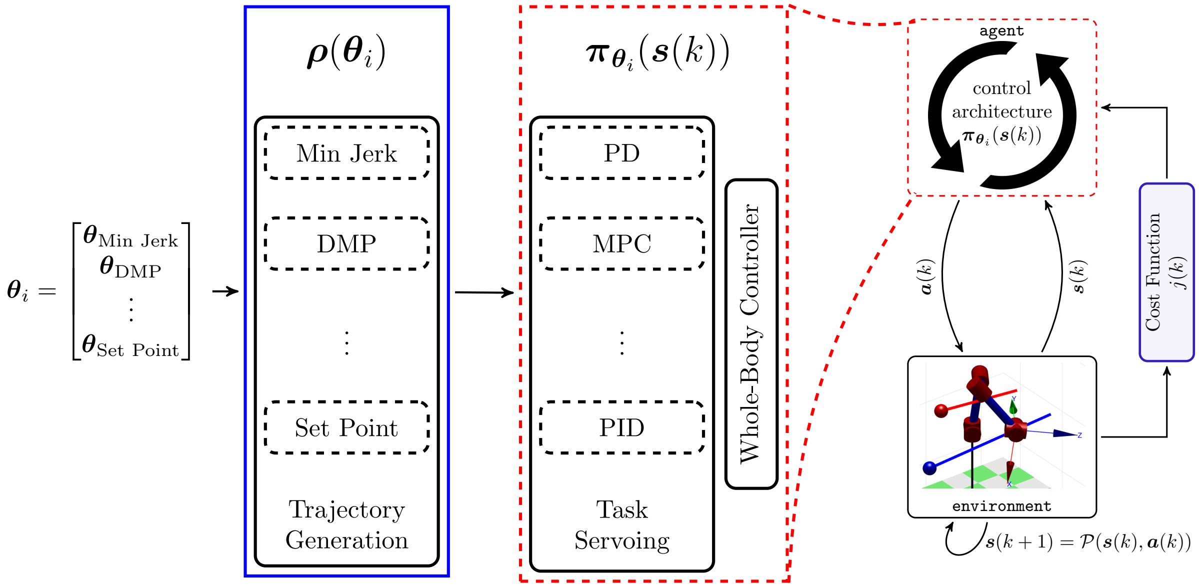

$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{\theta}_{\text{Min Jerk}} \\ \boldsymbol{\theta}_{\text{DMP}} \\ \vdots \\ \boldsymbol{\theta}_{\text{Set Point}} \end{bmatrix}$
$\boldsymbol{\rho}(\boldsymbol{\theta}_i)$
Min Jerk
DMP
⋮
Set Point
Trajectory Generation
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
PD
MPC
⋮
PID
Task Servoing
Whole-Body Controller
agent
control architecture
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
$\boldsymbol{a}(k)$
$\boldsymbol{s}(k)$
Cost Function
$j(k)$
environment
$\boldsymbol{s}(k+1) = \mathcal{P}(\boldsymbol{s}(k), \boldsymbol{a}(k))$

Figure 5.2 – A visual description of the MFPS policy formulation and parameterization. The parameterized policy generation function is the collection of task trajectory generators. Their parameters are the policy parameters. The policies then contain the task servoing and whole-body control layers of the model-based control hierarchy. This policy determines the actions to take, (torques), given the current state of the environment.

5.3 Policy Parameterizations

Given the high dimensionality of the system's states and actions, it is necessary to seek a lower dimensional representation, which can be more easily manipulated. This can be accomplished by using a parameterized function to generate the policy,

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{s}(k)) = \boldsymbol{\rho}(\boldsymbol{\theta}), \tag{5.2}$$

---

[1] Here, control policies are deterministic and time-dependent, unless otherwise indicated.

where $\boldsymbol{\theta}$ is a vector of parameters which dictate the resulting policy, $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. By design, the task reference values provided by the task trajectories dictate the resulting control policy. Therefore, any valid trajectory parameterization is also a valid policy parameterization. This has the advantage of easily being integrated into existing control schemes.

A variety of representations can be employed (e.g. splines, polynomials, etc.) for producing task trajectories. It is the ensemble of task trajectory generators, and their parameters, $\boldsymbol{\theta}_i$, which then constitute the policy parameterization, $\boldsymbol{\rho}(\boldsymbol{\theta})$ in (5.2), and,

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_{n_{\text{task}}} \end{bmatrix}.. \tag{5.3}$$

The $\boldsymbol{\rho}(\boldsymbol{\theta})$ function is therefore simply the set of task trajectory generators, and the reference output of these trajectories dictates the policy executed by the robot. The choice of trajectory generation scheme has a large impact on the manner in which $\boldsymbol{\rho}(\boldsymbol{\theta})$ behaves when components of $\boldsymbol{\theta}$ are modified, and they can roughly be broken up into two main categories:

1. Parameterizations learned from existing trajectory demonstrations.
2. Parameterizations based on task keyframes or waypoints.

Here we look at both categories, and detail a few specific methods applied to the problem of task compatibility and feasibility.

5.3.1 Learning From Demonstration

When one or more task trajectories are provided a priori, it is necessary to parameterize these preexisting trajectories in order to parameterize the policy. Associating a parameterized function with one or more trajectories is typically accomplished using regression techniques, where the trajectory data is approximated by a parametric or non-parametric function [Atkeson and Schaal, 1997]. Despite their name, non-parametric functions, such as Gaussian Processes (GPs), do have parameters but the number of those parameters is unbounded. These functions must project the trajectory (typically a non-linear function in time) into a higher dimensional space where linear regression can be used to approximate the projected data, or trajectory. Any number of functions can be used, but polynomials and most prominently, linear combinations of non-linear kernel functions, are commonly implemented [Billard et al., 2008; Argall et al., 2009]. Many different kernel functions exist, but all consist of meta-parameters which are found through regression and optimization techniques. Once this function is learned, it is generally coupled with a servoing controller to ensure that the resultant trajectory is stable. In recent years, Dynamical Movement Primitives (DMPs) and their variants have become a popular way to learn trajectories from demonstration [Ijspeert et al., 2013].

Dynamic Movement Primitives

DMPs are used to learn movements or trajectories from demonstration. They are based on point or limit cycle attractor landscapes augmented with a learned forcing term to guide the trajectory from start to goal, and typically some additional coupling terms to regulate their temporal evolution. In the context of humanoid robotics, DMPs present an interesting tool with which to learn, modify and store various routine movements or tasks. In Sec. A.4 of the Appendix, a DMP formulation based on [Kulvicius et al., 2012] and [Stulp, 2014] is detailed. The first two lines of the DMP presented in (A.1),

$$\begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{z}} \\ \vdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}/\tau \\ ((K(\boldsymbol{y}^{\gamma} - \boldsymbol{y}) - D\boldsymbol{z}) + v \cdot \boldsymbol{f}(x, \boldsymbol{\theta}))/\tau \\ \vdots \end{bmatrix}, \tag{5.4}$$

are the most important because they show the primary elements of the DMP. The most important of these terms is the forcing term, $\boldsymbol{f}(x, \boldsymbol{\theta})$, which parameterizes the trajectory. This forcing term is a non-linear kernel based function, whose meta-parameters, $\boldsymbol{\theta}$, are determined through regression fitting a set of similar trajectories which have been demonstrated by an expert. These kernel meta-parameters, which in this case are the affine slopes and offsets, $\boldsymbol{a}$ and $\boldsymbol{b}$, as well as the Gaussian centers and widths, $\boldsymbol{c}$ and $\boldsymbol{\sigma^2}$,

$$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \\ \boldsymbol{\sigma^2} \end{bmatrix}, \tag{5.5}$$

can then be used as policy parameters in $\boldsymbol{\rho}(\boldsymbol{\theta})$. The number of meta-parameters, $\boldsymbol{\theta}_i$, depends on the number of kernels used in $\boldsymbol{f}(x, \boldsymbol{\theta})$, which can be chosen through heuristics or with automated methods.

5.3.2 Waypoint Based Methods

If a trajectory is not provided for the task ahead of time, then it is necessary to generate one from the goals of the task. These goals are specified most commonly through keyframes, or *waypoints*, which represent task coordinates of particular importance. These coordinates are specific to the task being controlled and may be Cartesian poses, $\boldsymbol{\xi}$, joint angles, $\boldsymbol{q}$, wrenches, ${}^{e}\boldsymbol{\omega}$, and so on. Inherent in these waypoints are the overall objectives of some task; e.g., for a point-to-point reaching task, the waypoints are given by the start and goal states of the reach. This is the manner in which tasks are generally specified by an expert operator in human planning [Yanco et al., 2015].

In this study, it is assumed that the objective of any trajectory is to pass through the waypoints or else they should not be specified. For instance, a single waypoint for an operational-space acceleration task, $\boldsymbol{\lambda}_i$, with 6 DoF in this case, ξ, is given by,

$$\boldsymbol{\lambda}_i = \boldsymbol{\xi}_i = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \dots \\ \xi_6 \end{bmatrix}_i \tag{5.6}$$

while a set of n_λ waypoints is denoted,

$$\Lambda = \begin{bmatrix} \boldsymbol{\lambda}_1 & \boldsymbol{\lambda}_2 & \dots & \boldsymbol{\lambda}_{n_\lambda} \end{bmatrix}. \tag{5.7}$$

For all trajectory generation schemes, an associated time law is either generated or required. Thus, it is sometimes necessary to associate a corresponding time step, t_i^λ, to each waypoint, $\boldsymbol{\lambda}_i$. This can be done by hand or via some heuristic method, and should provide the waypoint times in the vector,

$$\boldsymbol{t}^\lambda = \begin{bmatrix} t_1^\lambda & t_2^\lambda & \dots & t_{n_\lambda}^\lambda \end{bmatrix}. \tag{5.8}$$

From Λ, and possibly $\boldsymbol{t}^\lambda$, a trajectory should be generated, which outputs,

$$\begin{bmatrix} \boldsymbol{\xi}^{\text{ref}} & \dot{\boldsymbol{\xi}}^{\text{ref}} & \ddot{\boldsymbol{\xi}}^{\text{ref}} \end{bmatrix} \tag{5.9}$$

at each time, t, from t_1^λ to $t_{n_\lambda}^\lambda$.

As proposed by [Lober et al., 2016], for policy parameterization, the waypoints and their times can be used,

$$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{\lambda}_1^\top & \boldsymbol{\lambda}_2^\top & \dots & \boldsymbol{\lambda}_{n_\lambda}^\top & t_1^\lambda & t_2^\lambda & \dots & t_{n_\lambda}^\lambda \end{bmatrix}^\top. \tag{5.10}$$

If one or more waypoints and times are fixed, then a subset of $\boldsymbol{\theta}_i$ can be used in the policy parameters. Each case depends on the task being executed.

Gaussian Process Trajectories

Given Λ and $\boldsymbol{t}^\lambda$, a trajectory generator should be able to produce a smooth (twice differentiable) trajectory which respects the given times. The output at each time step is the task reference values. For this, generators such as splines, polynomials, and blended linear motions, are all good candidates. Though, to go a bit further, it would be useful if the generator produced not only a trajectory, but bounds on the trajectory. That is to say, at the specified waypoints, near perfect accuracy is desired or else they would not have been specified; however, between waypoints, having some margins for error in the trajectory could be useful. One example would be to provide bounds for the NMPC problem described in Sec. 4.4. These margins for error then represent spatial variance in the trajectory.

Trajectory variance is typically calculated from multiple demonstrations of the same movement [Calinon et al., 2010; Kormushev et al., 2010]. Unfortunately, demonstration data is not always available, and it is advantageous to be able to compute task variance with only waypoint data. For this purpose, there are no "off the shelf" solutions. However, inspiration can drawn from the field of *Gaussian Process* (GP) regression. GPs, are widely used in machine learning for modeling complex data sets [Rasmussen and Williams, 2006]. They belong to the family of kernel based supervised learning techniques and are used to stochastically model relationships between input and output data. Depending on the kernel function used to compute the GP, they are mathematically similar to b-splines, but with the added advantage of providing an associated prediction variance for each predicted output mean. The details of GP regression are provided in Sec. A.5 of the Appendix.

To exploit GPs for the purpose of trajectory generation, and create a non-parametric stochastic trajectory generator, one can simply substitute the waypoints, Λ, for $\boldsymbol{b}$ and their time steps, $\boldsymbol{t}^\lambda$, for $\boldsymbol{a}$ in the GP equations from (A.14). This process is originally described by [Lober et al., 2015], where firstly, $\boldsymbol{t}^\lambda$ is determined using a conservative maximum velocity, $\dot{\xi}_{max}$, between waypoints and by computing,

$$t^\theta_{i+1} = t^\theta_i + \frac{\|\boldsymbol{\lambda}_{i+1} - \boldsymbol{\lambda}_i\|_2}{\dot{\xi}_{max}} \; \forall i \in 2, 3, \ldots, n_\lambda, \tag{5.11}$$

with $t^\theta_1 = 0.0s$. Evaluating the GP equations at each new time step, t, using Λ and $\boldsymbol{t}^\lambda$, outputs a trajectory mean, $\boldsymbol{\xi}_\mu(t)$, and variance, $\boldsymbol{\xi}^2_\sigma(t)$,

$$\begin{bmatrix} \boldsymbol{\xi}_\mu(t) \\ \boldsymbol{\xi}_{\sigma^2}(t) \end{bmatrix} = \begin{bmatrix} K_* K^{-1} \Lambda^T \\ K_{**} - K_* K^{-1} K_*^T \end{bmatrix} = \mathcal{GP}(t, \boldsymbol{t}^\lambda, \Lambda) \; . \tag{5.12}$$

With a *Gaussian Process Trajectory*, or *GPT*, the input, t, is unidimensional, while the output, $\boldsymbol{\xi}$, may be multidimensional. The variable dependencies for terms, $K(\boldsymbol{t}^\lambda)$, $K_*(t, \boldsymbol{t}^\lambda)$, and $K_{**}(t)$ have been omitted from (5.12) for clarity. The velocity and acceleration terms, $\dot{\boldsymbol{\xi}}_\mu(t)$ and $\ddot{\boldsymbol{\xi}}_\mu(t)$, are calculated using the finite difference method.

$$\dot{\boldsymbol{\xi}}_\mu(k) = \frac{\boldsymbol{\xi}_\mu(k) - \boldsymbol{\xi}_\mu(k-1)}{h} \; , \tag{5.13}$$

and

$$\ddot{\boldsymbol{\xi}}_\mu(k) = \frac{\dot{\boldsymbol{\xi}}_\mu(k) - \dot{\boldsymbol{\xi}}_\mu(k-1)}{h} \; . \tag{5.14}$$

In (5.13) and (5.14), h is the control period. To ensure that the velocity goes to zero and smooth out the start and stop of the motion, two "clamping" waypoints can be added to the movement at the beginning and end of the GPT.

Selection of the kernel meta-parameters, σ_k and l_k, plays a crucial role in the form of the GPT generated by (5.12). One option is to hand tune the parameters to attain various smoothnesses and maximum variance properties. For instance, a larger value of l_k generates smoother GPT profiles, but less variation in variance between waypoints, and the maximum covariance, σ_k, only affects the variance term, $\boldsymbol{\xi}^2_\sigma$, and not $\boldsymbol{\xi}_\mu$. As such, it may be useful to employ different l_k for the calculation of the mean and variance of the GPT. However, such tuning may render the GP stochastically invalid, i.e. there is no real correlation with the provided training data, Λ and $\boldsymbol{t}^{\lambda}$. Another option is to find the meta-parameters through maximum likelihood estimation or maximum a posteriori estimation.

Using these simple equations, a stochastic trajectory generator can be produced which provides a mean trajectory of task references and variances associated with them. Because GPTs provide both mean and variance values for each DoF of a task trajectory based on only a set of waypoints, there is now a way to both parameterize the policy and its variance.

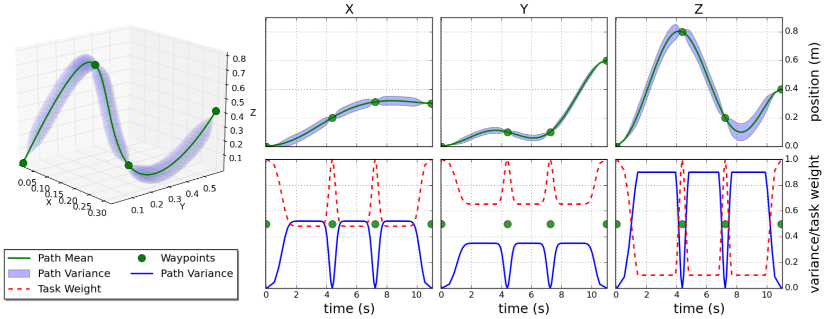


Figure 5.3 – An example of a 3D task trajectory with variance. This figure shows how variance can be computed given a Gaussian Process Trajectory, or GPT, then mapped to the weights of the individual DoF of the task.

Stochastic Policies with GPTs

Task trajectories parameterized by GPTs possess both mean trajectory values, $\boldsymbol{\xi}_{\mu}(t)$, and spatial variances associated with those means, $\boldsymbol{\xi}_{\sigma^2}(t)$, at each time, t. In robotics, this variance information can be used for purposes such as, bounding a stochastic search space as shown by [Kang and Park, 2014], calculating task gains as demonstrated by [Buchli et al., 2011], [Calinon and Li, 2012], and [Kormushev et al., 2010], or blending tasks [Paraschos et al., 2013]. To exploit this variance within the policy, one restriction must be made on the underlying controller — it must use weighted prioritization (see Sec. 2.4.2)[2]. In [Lober et al., 2015] it is shown that the task trajectory variance can be used to infer the relative importance of the tasks at each time step. This constrains the model-free formulation, but opens up the possibility of using *stochastic policies*, $\boldsymbol{\pi}(\boldsymbol{a}(k)|\boldsymbol{s}(k))$.

High variance values indicate low imperative for following the trajectory precisely, while low values express the opposite. It is shown that this information can be exploited to guide the modulation of task weights in the controller, rendering the tasks more robust to incompatibilities and infeasibilities. This so-called *variance modulated prioritization*, is accomplished using a simple linear map between task trajectory variance and task weight, or priority,

$$\boldsymbol{w}_i(t) = \frac{\mathbf{1} - \boldsymbol{\xi}_{\sigma^2}(t)}{\beta}, \tag{5.15}$$

where, $\boldsymbol{w}_i(t)$ is the weight vector for task f^{task} at time t, and $\mathbf{1}$ is a vector of ones with the same dimension as $\boldsymbol{\xi}_{\sigma^2}$. When the variance of the movement, $\boldsymbol{\xi}_{\sigma^2}(t)$, is high, its value is close to 1 and the weight/importance of the task diminishes. As $\boldsymbol{\xi}_{\sigma^2}(t)$ approaches 0, the importance of the task increases towards its maximum. The factor β allows one to scale the overall importance of the task relative to the other tasks, while still maintaining variability. A $\beta > 1$ means the variable weight task is less important than the other tasks, while $\beta < 1$ the inverse. This is useful when combining uncritical tasks with safety-critical tasks such as balancing; however, it does not guarantee that the safety-critical task will go unperturbed.

---

[2] An alternative approach, which has been explored by Buchli et al. [2011] and Calinon et al. [2014], uses the task variance for state-feedback gain scheduling. This approach would not place any restriction on the type of whole-body controller used.

In practice, if the variance is too close to 1.0, the weight of the task becomes infinitesimal and the controller no longer executes it. In order to avoid such behavior, the maximum $\boldsymbol{\xi}_{\sigma^2}(t)$ can be bounded at a value just less than 1.0 (e.g., 0.99 is used here). Figure 5.3 shows an example of a 3-dimensional Cartesian trajectory generated by a GPT, with the evolution of the mapped weights.

If the tasks are all compatible and feasible, the weight of the tasks does not matter and the mean trajectory given by $\boldsymbol{\xi}_{\mu}(t)$, for all the tasks will be executed. However, if this is not the case, then incompatibilities and infeasibilities will cause the task to deviate from $\boldsymbol{\xi}_{\mu}(t)$. The amount of deviation depends on the variance, $\boldsymbol{\xi}_{\sigma^2}(t)$, at the instant of incompatibility/infeasibility. The result is a policy which depends not only on the parameters $\boldsymbol{\theta}$ but also on the incompatibilities and infeasibilities generated by $\boldsymbol{\theta}$. As a result, for the same policy parameters, the policy may produce different actions depending on the existence and timing of task incompatibilities and infeasibilities. Thus the term, stochastic policy.

Time-Optimal Trajectories

As is the case with GPTs, determining the waypoint time steps, $\boldsymbol{t}^{\lambda}$, is not always straightforward and greatly affects the dynamics of the trajectory. Even through the use of conservative heuristics as in Sec. 5.3.2, the resulting motion may contain infeasible accelerations and decelerations. Consequently, a trajectory generator is required for cases where only Λ and maximum velocity and acceleration limits are given for the task. In the work of [Kunz and Stilman, 2012], a *Time-Optimal Trajectory*, or *TOT*, generator, which only requires waypoints and limits on the velocity and acceleration, is presented. The resulting TOT consists of straight line paths between waypoints, with circular blends. The time law determined by the algorithm provides the fastest trajectory through the waypoints which respects the velocity and acceleration limits. Thus the TOT duration, t_{n_λ}, depends on the velocity and acceleration limits imposed on the movement. For the policy parameterization, the waypoint times, $\boldsymbol{t}^{\lambda}$ are no longer variables which can be manipulated. Therefore, only the waypoints which are deemed "adjustable" are included in the policy parameters,

$$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_{n_\lambda} \end{bmatrix}. \tag{5.16}$$

5.4 Policy Rollouts

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

$$\{\mathcal{S}, \mathcal{A}\} = \text{rollout}(\boldsymbol{\pi_\theta}), \tag{5.17}$$

where $\mathcal{S}$ and $\mathcal{A}$ are the sets of the states and actions over the entire rollout. This implies that the full control architecture described by the parameterized policy, as shown in Fig. 5.2, is employed until the task execution is complete, meaning that the execution must occur in a finite amount of time and should be finished in the duration, $t_{\boldsymbol{\pi}}$, dictated by the policy $\boldsymbol{\pi_\theta}$. This duration, $t_{\boldsymbol{\pi}}$, can be based on the largest task trajectory duration, or dictated by some other high-level means such as the time when a motion is completed. In the later case, it is possible that $\boldsymbol{\pi_\theta}$ will result in a failure (falling for instance) and the overall motion will never be completed. The policy rollouts are therefore assigned a maximum execution time, $t_{\boldsymbol{\pi}}^{\max} > t_{\boldsymbol{\pi}}$, to allow for possible delays in the task execution but to avoid recording failed rollouts indefinitely. Here, we arbitrarily select,

$$t_{\boldsymbol{\pi}}^{\max} = 1.5 t_{\boldsymbol{\pi}}. \tag{5.18}$$

5.5 Task Compatibility and Feasibility Cost Functions

In MFPS, the cost function tells an outside observer, or oracle, how to interpret the result of an action given a set of states, i.e. the policy's performance. The objective here is to develop cost functions which allow the incompatibility and/or infeasibility of the tasks, to be measured. Moving to a model-free approach means that task compatibility and feasibility should no longer be calculated using model-based metrics, but measured using rollout data. Ultimately, we are concerned with minimizing the side-effects of task incompatibilities and infeasibilities, which have a negative impact on the whole-body motion. With this in mind, three cost functions are developed, which capture the key side-effects to be measured. These cost functions are derived from performance measures, which are commonly used in optimal control, and calculated a posteriori on the rollout data determined in (5.17).

5.5.1 Tracking Cost

Using the state information $\mathcal{S}$, one can determine how each task evolved over the course of a single rollout. For any task, the best case scenario is that its objective function is perfectly minimized and therefore zero. This would indicate that the robot perfectly follows the task reference. Thus, any error in the position tracking reflects an imperfect optimization of the task error and consequently a task incompatibility or infeasibility. Using this concept, a *Tracking (T) Cost* is defined as,

$$j^{\mathrm{T}} = \sum_{k=0}^{N} \|\boldsymbol{\epsilon}(k)\|_2^2, \tag{5.19}$$

where $\boldsymbol{\epsilon}(k)$, is the task position tracking error at time k, and is summed over the total number of time steps, N. The actual total duration of the rollout is defined as $t_{\boldsymbol{\pi}}^{\mathrm{end}} = Nh$, where h is the control sampling period, and $t_{\boldsymbol{\pi}} \leq t_{\boldsymbol{\pi}}^{\mathrm{end}} \leq t_{\boldsymbol{\pi}}^{\mathrm{max}}$. Although (5.19) is applied to acceleration tasks, tracking error can be computed for wrench and torque tasks as well.

If a task error is perfectly minimized by the controller, then the tracking error is zero, meaning that the robot perfectly executes $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. This would indicate perfect compatibility and feasibility. However, if (5.19) is greater than zero, which is likely, then some incompatibilities and/or infeasibilities are affecting the task performance. In (5.19), only one task is presented. If multiple tasks are active simultaneously, then this cost can be calculated for each, i.e. $j_i^{\mathrm{T}}\ \forall i \in 1, 2, \ldots, n_{\mathrm{task}}$.

5.5.2 Goal Cost

While the tracking cost in (5.19), measures how well each task performed on a whole, it does not indicate directly if each task attains its goal reference value. This goal reference comes from the task trajectory and is generally the last waypoint or reference position in the trajectory. We make the assumption that the ultimate objective of any point-to-point trajectory is to reach its goal reference. This assumption is based on the idea that any non-cyclic trajectory is designed to move the robot from one point to another on a specific path. In some cases, only the path matters and the goal is simply part of it, e.g. welding. In the majority of other cases, the goal is the only state that matters and must be attained, e.g. reaching for an object.

The criteria for whether or not a task has "attained" its goal reference or not are somewhat vague, as precision requirements vary on a task by task basis. Therefore, a continuous cost is developed to attribute increasing costs the further the task is from its goal reference. Thus a *Goal (G) Cost* is developed as,

$$j^{\mathrm{G}} = \sum_{k=0}^{N} \frac{kh}{t_{n_\lambda}} \|\boldsymbol{\epsilon}^{\mathrm{goal}}(k)\|_2^2. \tag{5.20}$$

The error term, $\boldsymbol{\epsilon}^{\mathrm{goal}}(k)$, is the difference between the current task position and its last waypoint $\boldsymbol{\lambda}_{n_\lambda}$ at the k^{th} time step. In lieu of a final waypoint, for trajectories based on DMPs for example (see Sec. 5.3.1), the final position reference from the task trajectory can be used. The error is multiplied by a penalizing factor $\frac{kh}{t_{n_\lambda}}$, which increases linearly from 0.0 with time. Again, t_{n_λ} is the final waypoint time step, or the expected task trajectory duration. If a task does not have a t_{n_λ} because its references are static (e.g. a set point), then the rollout duration, $t_{\boldsymbol{\pi}}^{\text{end}}$, can be used instead of t_{n_λ}. This means that the distance to the goal waypoint becomes more important as time elapses. For $kh < t_{n_\lambda}$, the error is discounted by a factor < 1.0, while for $kh \geq t_{n_\lambda}$ the error is increased by a factor ≥ 1.0. Thus as the execution of the task, exceeds its planned duration, t_{n_λ}, the goal error is increased.

The goal cost described by (5.20) can only be zero if the start and goal states are the same, i.e. the task should not move. Under these circumstances, it may be possible to obtain "perfect" task performance in the sense that there is zero goal cost. This would also indicate perfect task compatibility and feasibility. However, for tasks where the start and goal states are not the same, this goal is unattainable because for $k > 1$ the error value will be non-zero, and since this value accumulates, the goal cost will never be zero for the rollout. Nevertheless, if a task does reach its goal, the cost will go to zero at that particular time step, and the sum will not increase. Such behavior would indicate that the goal has been attained. On the other hand, if the goal is never attained, then the cost will continue to grow over the course of the rollout.

5.5.3 Energy Cost

In optimal control, one key performance criterion is the energy optimality of a particular control policy. Since many policies may successfully accomplish a particular objective, the issue then becomes determining which does it most efficiently. Given the redundancy of humanoid robots, it is possible that many whole-body motions exist for the same set of tasks to be performed. Out of these possibilities, one can determine the most energetically optimal motion by minimizing the actions, $\boldsymbol{a}$ (i.e. the control inputs, $\boldsymbol{\tau}$) using an *Energy (E) Cost*,

$$j^{\text{E}} = \sum_{k=0}^{N} \|\boldsymbol{\tau}(k)\|_2^2. \tag{5.21}$$

In terms of task compatibility and feasibility, measuring (5.21) alone does not provide much useful information on the task performance. However, between two policy rollouts, if one compares their energy costs and sees a dramatic ($> \times 2$) increase in energy usage, then it is highly likely that the motion has failed or the task(s) are incompatible and/or infeasible. This can be deduced based on two observations. Firstly, failures typically cause instabilities in the controller or the dynamics and these will most often saturate the actuators as the controller attempts to return the robot to a dynamically stable state. Secondly, when tasks are incompatible and/or infeasible, they generally accumulate large errors at the servoing level and these large errors generate large reference terms, which in turn cause the controller to use large torques to compensate. The energy cost is therefore a useful policy comparison measure between two or more rollouts.

5.6 Combining Cost Functions

Having developed three cost functions which capture various aspects of task performance, and therefore, task compatibility and feasibility, it remains to be seen how they should be applied. Both the goal and tracking costs are specific to individual tasks and indicate how well tasks achieved their objectives. The energy cost, on the other hand, is a global indicator. How these are combined depends on the problem at hand and the tasks which are being executed.

5.6.1 Task Performance Cost

Both the tracking and goal costs provide different pieces of information about the effects of incompatibilities and infeasibilities on a single task. The tracking cost from (5.19) shows how well the task was executed on a whole, without regards for which portions of the task trajectory were more successfully tracked than others. The goal cost from (5.20) focuses on the how well the primary objective of the task, i.e. attain the goal task state, was achieved, it does not however, indicate how well the task followed its trajectory. To determine the overall performance of a single task given its rollout data, the tracking and goal costs must be combined,

$$j_i^{\mathrm{TP}} = j_i^{\mathrm{T}} + j_i^{\mathrm{G}}, \tag{5.22}$$

to produce a *Task Performance (TP) Cost*. This cost is a measure of how poorly a single task is performed. If the cost is near zero, then the performance is near perfect, if the cost is large, then the performance is lackluster. With (5.22), one can evaluate the effects of incompatibility and infeasibility on a single task. To measure the cumulative performance degradation of the tasks, the individual j_i^{TP} can be summed,

$$j^{\mathrm{TP}} = \sum_{i=1}^{n_{\mathrm{task}}} w_i j_i^{\mathrm{TP}}, \tag{5.23}$$

where w_i is the task weight if this information is available or applicable[3], otherwise, $w_i = 1.0$. The *Total Task Performance Cost*, j^{TP}, shows how well the overall task set performed as a weighted sum of the individual task performance costs. In (5.23), all the tasks are used in the computation of j^{TP}, however, a subset of the tasks can also be used.

5.6.2 Total Performance Cost

Determining the total *Performance (P) Cost* of a policy, $\boldsymbol{\pi_\theta}$, is then accomplished by summing the total task performance cost, (5.23), and the energy cost, (5.21),

$$j^{\mathrm{P}} = \frac{j^{\mathrm{TP}} + \varphi j^{\mathrm{E}}}{t_{\boldsymbol{\pi}}^{\mathrm{end}}}, \tag{5.24}$$

where j^{P} is the performance cost, and φ is an energy scaling factor. The scaling term, φ is necessary because torque vector norms are typically numerically large compared to error norms at the task level. If not scaled, j^{E} would dominate any j^{TP}. Because policies may differ in durations, and j^{TP} and j^{E} are integrated over these durations, the sum of these two costs is averaged by the total duration of the rollout, $t_{\boldsymbol{\pi}}^{\mathrm{end}}$.

Scaling The Performance Cost Over Rollouts

The performance cost, (5.24), of the policy, $\boldsymbol{\pi_\theta}$, provides an estimate of how well the whole-body behavior was accomplished and therefore a quantitative view of the incompatibility or infeasibility of the task set. However, this scalar estimate has no absolute significance on its own. There is no threshold value for determining analytically if $\boldsymbol{\pi_\theta}$ was successful in a high-level sense. Given this ambiguity, each policy must be evaluated relative to the original policy in order to see if any improvement in performance has been made. Taking the j_0^{P} to be the performance cost of the original policy, $\boldsymbol{\pi}_{\boldsymbol{\theta}_0}$, all $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$ are scaled using j_0^{P},

$$\hat{j}_i^{\mathrm{P}} = \frac{j_i^{\mathrm{P}}}{j_0^{\mathrm{P}}}, \tag{5.25}$$

where i indicates the rollout number. This means that the initial, $\boldsymbol{\pi}_{\boldsymbol{\theta}_0}$, has a feasibility cost equal to 1.0 and any $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$ which produces a $\hat{j}_i^{\mathrm{P}} < 1.0$ represents an improvement in task performance, and vice versa for $\hat{j}_i^{\mathrm{P}} > 1.0$.

---

[3] If hierarchies are used, then one can choose to approximate the levels with weights, or not weight the sum of (5.22) at all.

5.7 Task Compatibility and Feasibility Maximization (TCFM)

With the model-free formulation, task parameterizations, and performance cost functions, we can now develop a MFPS algorithm for optimizing policy parameters to minimize the performance cost(s). By minimizing these costs, the compatibility and/or feasibility of the task(s) should be maximized.

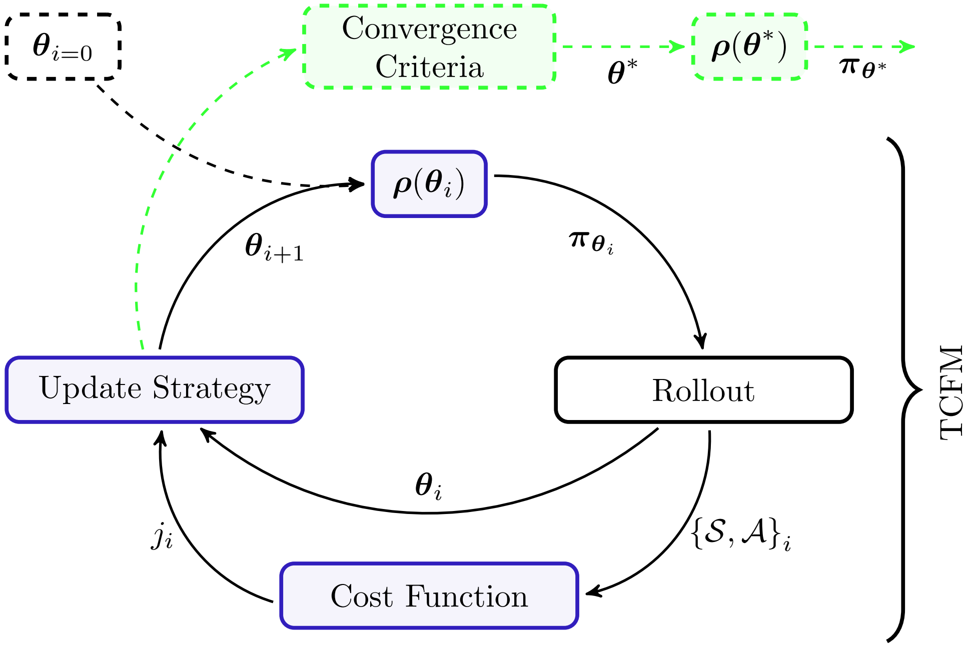

$\boldsymbol{\theta}_{i=0}$
Convergence Criteria
$\boldsymbol{\theta}^*$
$\boldsymbol{\rho}(\boldsymbol{\theta}^*)$
$\boldsymbol{\pi}_{\boldsymbol{\theta}^*}$
$\boldsymbol{\rho}(\boldsymbol{\theta}_i)$
$\boldsymbol{\theta}_{i+1}$
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$
Update Strategy
Rollout
TCFM
$\boldsymbol{\theta}_i$
j_i
$\{\mathcal{S},\mathcal{A}\}_i$
Cost Function

Figure 5.4 – A depiction of the TCFM policy search cycle. The critical components in this cycle are the cost function, update strategy, and policy parameterization scheme. This cycle is described in Algorithm 3.

Starting from the initial $i = 0$ policy parameters, $\boldsymbol{\theta}_{i=0}$, the policy, $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$ is generated using the parameterized function, $\boldsymbol{\rho}(\boldsymbol{\theta}_i)$. Again, $\boldsymbol{\rho}(\boldsymbol{\theta}_i)$ is composed of the various task trajectory generators, and $\boldsymbol{\theta}_i$ are their "adjustable" parameters, which may affect one or more task trajectories. The generated policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$, is then rolled out and this produces the state and action data, $\{\mathcal{S},\mathcal{A}\}$. Given this data, a performance cost is calculated to evaluate the policy. The result is a scalar cost, j_i. Using j_i and $\boldsymbol{\theta}_i$, an update strategy then predicts the next policy parameters, $\boldsymbol{\theta}_{i+1}$, to test, which should minimize the policy cost. This process is known as *episodic learning* and is iterated until the update strategy converges, or a maximum number of iterations is exceeded.

Here, an effort is made to not explicitly give the type of cost function or update strategy used within the MFPS to show that these two components can be defined on a case to case basis. The result is a vector of optimal policy parameters, $\boldsymbol{\theta}^*$, used to generate an optimized policy, $\boldsymbol{\pi}^*_{\boldsymbol{\theta}}$, which minimizes the cost, and therefore maximizes task compatibility and/or feasibility. This *Task Compatibility and Feasibility Maximization*, or TCFM, algorithm is depicted in Fig. 5.4 and detailed in Algorithm 3. Three components remain to be determined, the cost function, the policy parameters, and the update strategy.

In this work, the concern is ensuring that tasks being executed on a whole-body controller are compatible with one another, and feasible according to the system constraints. Thus, either j^{TP} or j^{P} are used as the TCFM cost function. Of course, given the general nature of MFPS any other cost function could be used, such as distance walked, or height jumped, if the objective was different. The policy parameters which are used to modify the policies, also require close consideration. The challenge of determining

Algorithm 3 TCFM- Policy Search

```
Given initial policy parameters: $\boldsymbol{\theta}_{i=0}$.
do
    $\boldsymbol{\pi}_{\boldsymbol{\theta}_i} = \boldsymbol{\rho}(\boldsymbol{\theta}_i)$                        ▷ generate policy from parameters
    $\{\mathcal{S}, \mathcal{A}\}_i = \text{rollout}(\boldsymbol{\pi}_{\boldsymbol{\theta}_i})$                              ▷ rollout the policy
    $j_i = \text{cost}\left(\{\mathcal{S}, \mathcal{A}\}_i\right)$                         ▷ calculate the cost function
    $\boldsymbol{\theta}_{i+1} = \text{update strategy}\left(j_i, \boldsymbol{\theta}_i\right)$        ▷ determine new parameters to test
while not converged or $i <$ max iterations                  ▷ evaluate convergence criteria
return $\boldsymbol{\theta}^*$                                         ▷ return incumbent solution
$\boldsymbol{\pi}^*_{\boldsymbol{\theta}} = \boldsymbol{\rho}(\boldsymbol{\theta}^*)$                                   ▷ generate optimized policy
```

which parameters have a consequential impact on the policy, and therefore the whole-body motion, is explored in the following section. Similarly, the update strategy has only sets of parameters and their associated costs to work with and must use only this information to determine the next $\boldsymbol{\theta}_{i+1}$ to test. Doing this efficiently depends heavily on how the update strategy explores the parameter space. Update strategies are examined in Sec. 5.7.2.

5.7.1 Determining Policy Parameters

The first step of TCFM, is to determine which policy parameters, $\boldsymbol{\theta}$, can be modified, i.e. the "adjustable" parameters. For example, task trajectories are often parameterized by only starting and goal positions. Starting positions cannot be changed, because they are dictated by the current robot state. This leaves only the goal positions to be changed; however, goal positions generally represent the ultimate objective of the task, i.e. the reason the task exists is to move the robot to this particular goal position. According to this logic[4], the goal positions should not be modified either, leaving no policy parameters, with which the policy can be modified to decrease the performance cost. In such cases, one must determine a new parameterization for the task trajectories, which maintains the constraints of the starting and goal task states, but introduces "adjustable" parameters, which can be exploited to change the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. From this, there are two possible manners to proceed:

1. Use learning from demonstration to model the trajectory as a DMP, and use the DMP parameters to modify $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. See Sec. 5.3.1.

2. Use the start and goal waypoints gleaned from the original task trajectories and select other intermediate waypoints from these trajectories as "adjustable" waypoints, then use GPT, TOT, or any other parameterized trajectory generator, to reparameterize the task trajectories. See Sec. 5.3.2 and Sec. 5.3.2.

With these two methods of adding "adjustable" parameters, any set of task trajectories can be reparameterized. If the policy is being created from scratch, meaning that the task trajectory generators need to be selected, then it is simple enough to choose extra modifiable parameters from the very beginning. The challenge then is determining which tasks to use for modifying $\boldsymbol{\pi}_{\boldsymbol{\theta}}$.

The choice of which task parameters to optimize with TCFM is one of primordial importance, and unfortunately without a definitive answer. Ideally, all tasks would be parameterized to provide parameters with which to adjust $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, but practically speaking this introduces too many parameters, with too little impact on the performance. Despite this conundrum, one can nevertheless proceed fairly easily, by allowing any principal tasks to be modified. If the task set has one specific purpose, such as a reach, then another smart choice would be to simply modify the reaching task, leaving all others the same.

---

[4] There are cases where the goal position is unimportant or unspecified, namely rhythmic motions. We constrain the scope of this work to only look at point-to-point motions.

5.7.2 Update Strategy

The update strategy in TCFM is the component which determines the next policy parameters, $\boldsymbol{\theta}_{i+1}$, to test given a history of parameters and their costs,

$$\langle \Theta, \boldsymbol{j} \rangle = \left\langle \begin{bmatrix} \boldsymbol{\theta}_1^\top \\ \boldsymbol{\theta}_2^\top \\ \vdots \\ \boldsymbol{\theta}_i^\top \end{bmatrix}, \begin{bmatrix} j_1 \\ j_2 \\ \vdots \\ j_i \end{bmatrix} \right\rangle. \tag{5.26}$$

Using this data, the update strategy searches the parameter space to determine the $\boldsymbol{\theta}_{i+1}$, which best minimizes the cost. The choice of update strategy for the MFPS is a critical design decision and must be based on the application. Regardless of the update strategy, there are some common criteria, which must be met before selecting an update strategy. First and foremost, the relationship between the optimization variable, $\boldsymbol{\theta}$, and its cost, j, is non-convex and likely discontinuous. This is due to the complexity of the policy itself, which contains the task servoing and whole-body control layers of the control architecture. In addition to the policy complexity, each rollout consists of a full task set execution and therefore an integration of the non-linear rigid body dynamics, or transition dynamics, of the system with potentially changing contacts. This results in a mapping between $\boldsymbol{\theta}$ and j that is not bijective and cannot be properly described by a differentiable function. Accordingly, any update strategy which is selected must not depend on analytical cost function gradients. This limits the available update strategy choices to *derivative-free methods*.

In addition to the mathematical complexity of the parameter to cost mapping, a serious practical consideration in update strategy choice is the number of rollouts needed for the update strategy to converge. This is called the *sample efficiency* of the update strategy and is an important topic in robotics, because executing a rollout (evaluating policy parameters) requires either numerical integration of complex dynamics, or an actual execution of the tasks on a real robot. Performing rollouts in simulation can be time consuming for complex robots and scenarios, but performing rollouts on real robots can be dangerous. This is especially true when the robot is a humanoid and can lose balance. Thus, it behooves the MFPS designer to choose the most sample efficient update strategy available. Consequently, update strategies such as *random search*, which are sample inefficient, but simple to implement, are incompatible with real robotics applications. With these constraints, two categories of update strategies are explored.

Stochastic Optimization

Stochastic optimization is a set of optimization methods which model the underlying parameter, $\boldsymbol{\theta}$, to cost, j, mapping as a stochastic process where, $\boldsymbol{\theta}$ is treated as a random variable with a mean and covariance. As parameter and cost data, $\langle \Theta, \boldsymbol{j} \rangle$, are accumulated, the mean and covariance of $\boldsymbol{\theta}$ are updated. It is this update that varies between methods and determines the sample efficiency of stochastic optimization techniques. The update of the $\boldsymbol{\theta}$ mean and covariance is one of the most critical aspects of stochastic optimization and a number of methods exist to this end. Parameter updates are generally accomplished with either gradient estimation or probability weighted averaging [Stulp and Sigaud, 2012]. In this work, two update methods are explored.

PI$^{\text{BB}}$ The first, is known as *Policy Improvement though Black-Box optimization* algorithm, or PI$^{\text{BB}}$ [Stulp and Sigaud, 2013; Munzer et al., 2014]. In PI$^{\text{BB}}$, the probability, p_i, given to a particular sample, $\boldsymbol{\theta}_i$, is calculated as,

$$p_i = \exp\left(-\eta \frac{j_i - \min(\boldsymbol{j})}{\max(\boldsymbol{j}) - \min(\boldsymbol{j})}\right), \tag{5.27}$$

where η is an eliteness parameter which governs the trade-off between exploration and exploitation. Using these probabilities, the $\boldsymbol{\theta}$ mean update is calculated as,

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{p}^{\top}\Theta, \tag{5.28}$$

where Θ is the concatenation of the tested parameters (candidate solutions), as shown in (5.26), and $\boldsymbol{p}$ is the vector of their probabilities.

CMA-ES The second, more well known, method, known as *Covariance Matrix Adaptation - Evolutionary Strategy*, or CMA-ES, is developed by [Hansen, 2006], and similarly uses techniques tantamount to natural gradient descent. CMA-ES updates the mean and covariance of the parameter distribution to maximize the likelihood of $\langle \Theta, \boldsymbol{j} \rangle$. Additionally, the path taken in the parameter space by the direction changes of the covariance matrix is guided by a genetic algorithm, hence the Evolution Strategy. The CMA-ES update process is depicted in Fig. 5.5.

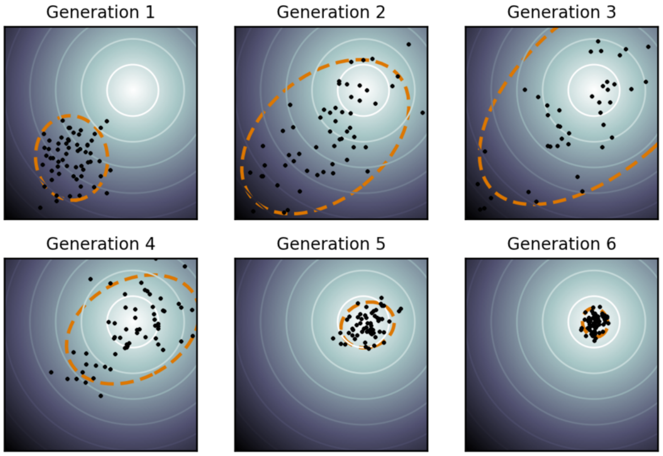


Figure 5.5 – A graphical representation of the CMA-ES policy update process. Here, the distribution of the parameter and cost data is used to randomly sample the solution space and the results are used to update the distribution in the direction of the natural gradient. Courtesy of `https://en.wikipedia.org/wiki/CMA-ES`.

Despite being more sample efficient than random methods, stochastic optimization techniques still generally require many rollouts to converge to an optimum. Furthermore, the optima found are typically local and within the vicinity (in parameter space) of the original parameters. Therefore stochastic optimization cannot ensure global optimality, but the local optima that are found are generally robust to perturbation [Uryasev and Pardalos, 2013]. As an alternative to stochastic optimization, more sample efficient, but complex update strategies can be used.

Bayesian Optimization

In humanoid robotics, rollouts are time consuming and dangerous, and consequently, sample efficiency is of the highest importance in TCFM. If the selected policy parameterizations are reasonably small in dimension (i.e. $\dim(\boldsymbol{\theta}) \leq 10$), then highly efficient update strategies such as *Bayesian Optimization*, or BO, can be used. BO derives its sample efficiency from explicitly modeling the latent parameter to cost mapping using a Gaussian Processes (GP), and then using this model, or *surrogate function*, to explore the parameter space. The actual minimization is performed on an *acquisition function* which combines the cost means and variances provided by the GP to balance exploitation with exploration [Brochu et al., 2010]. Supplemental details on BO can be found Sec. A.6 of the Appendix.

Because the GP is computationally simple to evaluate (if the problem size is small), fast but sample inefficient methods can be employed to minimize the acquisition function. Similarly to [Cox and John, 1992], a *Lower Confidence Bounds* (LCB) acquisition function is used because we are concerned with minimization,

$$\mathrm{LCB}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}, \boldsymbol{j}) = \hat{\boldsymbol{j}}_{\mu} - \sqrt{\upsilon}\hat{\boldsymbol{j}}_{\sigma^2}, \tag{5.29}$$

where $\hat{\boldsymbol{j}}_{\mu}$ and $\hat{\boldsymbol{j}}_{\sigma^2}$ are the predicted cost mean and variance according to the surrogate function, and the term υ is calculated using a variant of the no-regret formulation developed in [Srinivas et al., 2010],

$$\upsilon = 2\log\left(0.3 i^{\frac{5}{2}}\pi^2\right), \tag{5.30}$$

where, i, is the rollout number. The acquisition function can be minimized using any non-linear non-convex routines. Here, direct search and CMA-ES are used.

If the parameter search space is bounded then BO can provide a global optimum (using direct search methods), assuming the GP provides a sufficiently detailed model of the latent cost function. The incumbent solution is taken as the best parameter and cost observation from the rollouts, $\langle \boldsymbol{\theta}^*, j^* \rangle$; therefore, the optimization does not depend on the sequence in which the rollouts are performed. One drawback to BO is that it does not guarantee convergence in most cases; however, the following criterion can be used for any generic optimization problem and is exploited here,

$$\|\boldsymbol{\theta}_{\mathrm{new}} - \boldsymbol{\theta}_{\mathrm{best}}\|_2 \leq \gamma, \tag{5.31}$$

where $\boldsymbol{\theta}_{\mathrm{best}}$ are the best observed policy parameters with the lowest cost and γ is a meta-parameter which dictates the minimum threshold for convergence.

5.8 Conclusions

Using the MFPS formalism, a generic algorithm for maximizing task compatibility and feasibility has been developed. Three components present critical design choices for the algorithm. The first is the policy parameterization. How the policies are parameterized depends somewhat on the context of the whole-body motion, but regardless, this choice directly impacts how a change in the policy parameters, $\boldsymbol{\theta}$, affects the whole-body motion. This has a strong influence on the policy search, and leads to the second design choice: the update strategy. Ultimately, the goal is to find optimal policies in the fewest number of trials, but there is a trade-off between expressive and complex parameterizations, and the number of rollouts needed to learn policies. Thus, the selected update strategy depends largely on the number of policy parameters. This is explored in the following chapters.

Finally, the cost function must be chosen. The cost functions developed in this chapter, allow various aspects of task compatibility and feasibility to be measured indirectly by quantifying their side-effects. These costs are generic to any task-based controller and can be mixed together to assess the performance of one or more tasks, or the whole-body motion. A summary of the cost functions can be found in Sec. A.3 in the Appendix. If the proposed cost functions are effective measures of the performance of the whole-body motion, then the TCFM algorithm can be tested to determine if by minimizing these costs, the task compatibility and feasibility can be maximized. It now remains to prove that these costs do in fact quantify task compatibility and feasibility. To do so, the model-based metrics are first validated in Chapter 6 and these are in turn used to validate the model-free costs in Chapter 7.

Part IV

RESULTS

Chapter 6

Evaluation of Model-Based Metrics

It remains to be seen if model-based task compatibility and feasibility metrics developed in Chapters 3 and 4 are useful for real problems. To evaluate this, a simple 6 DoF manipulator, using the same whole-body control architecture described in Chapter 2, is put into controlled task compatibility and feasibility scenarios. Knowing ahead of time that the tasks in these scenarios are either infeasible or incompatible allows us to appraise the ability of the metrics to usefully describe the phenomena. Through three experiments, a subset of the metrics are singled out as useful indicators of task compatibility and feasibility.

6.1 General Experimental Setup

In the following sections, a series of experiments are presented which serve as benchmarks for the task compatibility and feasibility metrics, developed in Chapters 3 and 4. The first experiment explores the task compatibility metrics. Tasks which are designed to be maximally compatible and incompatible are used as controls to determine if the metrics accurately indicate compatibility. In the second experiment, task feasibility is explored. Similarly to the task compatibility tests, tasks which are known to be feasible and infeasible are applied to the whole-body controller and the metrics are evaluated. Finally, a more realistic experiment is studied, where the tasks go from being both compatible and feasible to being incompatible and infeasible, then back to being compatible and feasible, allowing the temporal nature of both task compatibility and feasibility to be explored. The metrics are evaluated on their ability to both quantitatively and qualitatively summarize the compatibility and feasibility of the tasks. For all experiments, the whole-body control problem (and numerical integration) is computed every 10ms and the task compatibility and feasibility metrics are computed every 100ms. Computation of all metrics takes approximately 2 seconds in Matlab® using a computer with an Intel® 2.4Ghz Core i7 processor and 32Gb of RAM.

For the experiments, a simplified model of the PUMA 560 robot provided by the Robotics Toolbox for Matlab from [Corke, 2011] is used. The PUMA 560 is a 6-DoF robot. The torque limits of the actuators are as follows:

$$\tau_1 = \pm 100 \text{ Nm} \quad \tau_2 = \pm 80 \text{ Nm} \quad \tau_3 = \pm 60 \text{ Nm} \quad \tau_4 = \pm 40 \text{ Nm} \quad \tau_5 = \pm 20 \text{ Nm} \quad \tau_6 = \pm 10 \text{ Nm}. \tag{6.1}$$

The lower and upper joint limits, in addition to some specific configurations used in the proceeding tests are provided by Table 6.1. The term $\boldsymbol{q}_N$ is the nominal posture for starting the robot, $\boldsymbol{q}_V$ is the vertical posture which puts each articulation in the middle of its joint range defined by $[\boldsymbol{q}_{\min}, \boldsymbol{q}_{\max}]$, and $\boldsymbol{q}_{IF}$ is an infeasible joint posture which is $\boldsymbol{q}_{\min} - 10°$. The $\boldsymbol{q}_V$ posture does not solicit the actuators and theoretically requires zero torque to maintain.

(in degrees)	q_1	q_2	q_3	q_4	q_5	q_6
$\boldsymbol{q}_{\min}$	-160	-45	-225	-170	-100	-266
$\boldsymbol{q}_{\max}$	160	225	45	170	100	266
$\boldsymbol{q}_N$	135	45	-180	0	45	0
$\boldsymbol{q}_V$	0	90	-90	0	0	0
$\boldsymbol{q}_{IF}$	-170	-55	-235	-180	-110	-276

Table 6.1 – The joint position limits and some specific configurations of the PUMA 560 robot simulation used here. The variable $\boldsymbol{q}_N$ is the nominal posture for starting the robot, $\boldsymbol{q}_V$ is the vertical posture which puts each articulation in the middle of its joint range defined by $\boldsymbol{q}_{\min}$ and $\boldsymbol{q}_{\max}$, and $\boldsymbol{q}_{IF}$ is an infeasible joint posture which is $\boldsymbol{q}_{\min} - 10°$.

6.2 Experimental Setup: Task Compatibility

In this first set of tests, the goal is to understand which task compatibility metrics best identify and quantify the compatibility of the task set. To study these concepts, a series of multi-task scenarios are developed, which simulate various compatibility issues. By design, the tasks are either compatible or incompatible, and the values provided by the metrics are interpreted. For the following task compatibility studies, three different acceleration tasks are used. The first is an end-effector (EE) position task ($f^{\ddot{\boldsymbol{\xi}}}_{\text{EE}}$), the second is an elbow position task ($f^{\ddot{\boldsymbol{\xi}}}_{\text{EL}}$), and the third is a joint position, or postural, task ($f^{\dot{\boldsymbol{\nu}}}_{\text{JP}}$). The elbow and EE tasks constrain 3 DoF and the postural task constrains 6 DoF. Accordingly, there is no kinematic redundancy left in the robot with the application of these tasks. However, the same cannot be said for redundancy in $\boldsymbol{\chi} \in \mathbb{R}^n$ which contains $\dot{\boldsymbol{\nu}} \in \mathbb{R}^6$ and $\boldsymbol{\tau} \in \mathbb{R}^6$, therefore, $n = 12$. There are no external contacts and thus no external wrenches, ${}^e\boldsymbol{\omega}$, in $\boldsymbol{\chi}$. A regularization term on $\boldsymbol{\chi}$ is also used with $w_0 = 1e - 5$. The task parameters are provided in Table 6.2.

Table 6.2 – Task set for the task compatibility experiments. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Cartesian	EE	3	1.0	10
Cartesian	Elbow	3	1.0	10
Joint Position	All DoF	6	0.0001	10

To study the effects of task compatibility, three cases are explored:

1. **Compatible Tasks**

   Given the initial starting posture (see Fig. 6.1a), each task state is determined and used as the task references values in the PD task servoing described by (2.52). This means that the tasks should keep the robot in its starting posture. Furthermore, because the task references are determined from task states, which are compatible by virtue of the fact that the robot is already in that posture, the tasks should be maximally compatible. This scenario therefore represents a control for the compatibility metrics. Any estimate of compatibility should indicate that it is at a maximum.

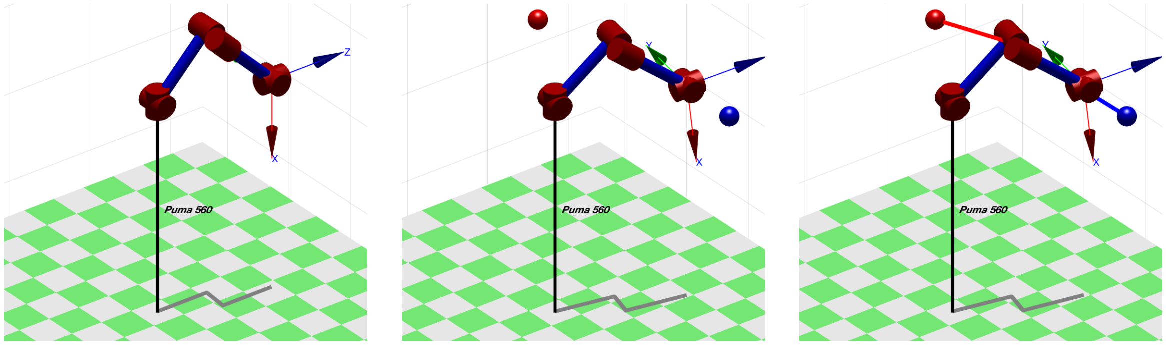
2. **Incompatible Tasks**

   In this case, the elbow and EE task references are set apart such that it is kinematically impossible for the robot to achieve both reference positions simultaneously. However, both the elbow and the EE tasks can be reached (kinematically feasible) when executed individually. In Fig. 6.1b the two reference task positions are indicated by the red and blue spheres for the elbow and EE tasks, respectively. These references are maximally incompatible from a kinematic standpoint. The postural task reference is set to the starting posture, $\boldsymbol{q}_N$.

3. **Incompatible Tasks Following Trajectories**

   In this final case, the same positions from case 2 are used, but instead of directly setting them as the reference values used in (2.52), they are used to generate straight line minimum jerk trajectories from the starting elbow and EE task positions to these desired positions. These trajectories are shown by the lines in Fig. 6.1c. Knowing that the final desired positions are incompatible, an evolution in the task compatibility should be observed as the tasks become more and more incompatible.

For each of the three cases, the robot starts its motion in the $\boldsymbol{q}_N$ posture and the movement is executed for 4 seconds.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 6.1 – The three task compatibility cases. The desired positions for the EE and elbow tasks are shown by the blue and red spheres respectively. For case 1, the starting positions are the same as the goal positions so they are not shown. For case 3, trajectories are used and shown by the blue and red lines.

6.3 Results: Task Compatibility

In this section, each of the three compatibility cases are assessed using the various task compatibility criteria developed in this work. First, the strict compatibility conditions are examined, then the compatibility metrics are computed and evaluated.

6.3.1 Strict Compatibility

To begin, the most restrictive qualification of strict compatibility is examined. Only case 1 should adhere to the definition of strict compatibility provided in Sec. 3.38. According to this definition, tasks are strictly compatible when $\operatorname{rank}(\tilde{E}) = \operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}])$. Therefore if,

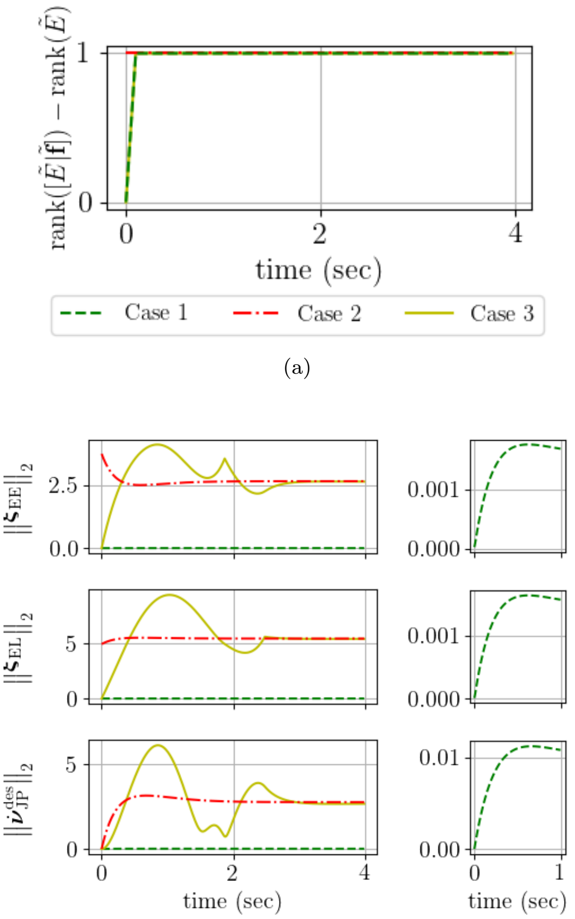
$$\operatorname{rank}([\tilde{E}|\tilde{\mathbf{f}}]) - \operatorname{rank}(\tilde{E}) > 0, \tag{6.2}$$

then the set of tasks is not strictly compatible. Again, $\tilde{E}$ and $\tilde{\mathbf{f}}$ are defined in (4.11). Plotting (6.2) for the three cases, Fig. 6.2a is obtained. At the very first time step both cases 1 and 3 follow to the notion of strict compatibility with $\mathrm{rank}(\tilde{E}) = \mathrm{rank}([\tilde{E}|\tilde{\mathbf{f}}])\ (= 7)\ < n\ (= 12)$. However, strict compatibility is lost after the first time steps because the desired task accelerations change the task objective functions.

Loss of Strict Compatibility

In Fig. 6.2b, the norms of the desired acceleration vectors for each task and for each case are plotted. It can be seen for case 2 that the initial desired accelerations start at non-zero values for the elbow and EE tasks and quickly rise to a non-zero value for the postural task. This explains why, for case 2, strict compatibility is lost at the first time step. Cases 1 and 3, however, start with zero desired acceleration for all tasks but slowly the desired accelerations begin to change. In case 3, this is due to the elbow and EE trajectories moving the robot towards incompatible positions, while in case 1, this is due to a settling of the controller equilibrium.

Controller Settling To be specific, any incompatibility between the objective functions will result in residual errors for each of the objective functions. These errors are then amplified by the task servoing gains. For case 1, this can be seen in the zoom in Fig. 6.2c, where the first second of accelerations are depicted. This causes the controller to seek a compromise between the task objectives based on their priorities and objective errors. Thus, the settling of the controller as the desired accelerations approach their equilibrium values asymptotically. This settling also indicates that the tasks are no longer strictly compatible. The question then is whether it is the changes in desired acceleration that cause the loss of strict compatibility or the loss of strict compatibility that causes the changes in acceleration?



(a)

(b)

(c)

Figure 6.2 – (a) rank difference for the linear system of equations defined by the task objective functions. Only cases 1 and 3 are initially strictly compatible, but both loose strict compatibility after the first few time steps do to changes in the desired accelerations. (b) desired acceleration norms for each task for each case. (c) zoom on the desired acceleration norms for task in case 1.

6.3.2 Compatibility Metrics

Having seen the sensitivity of the strict compatibility criterion, we now proceed with the task compatibility metrics.

Nuclear Norm Ratio

The first metric to be observed using these three test cases is the Nuclear norm Ratio, κ^{NR}. In Fig. 6.3, κ^{NR} is plotted for each case using the $\tilde{E}$ and $\tilde{\mathbf{f}}$ formulations from (4.11). According to the κ^{NR} values, case 1 maintains maximum compatibility throughout the motion execution. Case 2 on the other hand, has lower κ^{NR} values than case 1, indicating that its tasks are less compatible. Finally, the κ^{NR} values for case 3 evolve from perfectly compatible at the outset, to the same incompatibility as in case 2. This matches the controller response shown in Fig. 6.2b, and closely mimics the expected behavior of the three cases. Thus, the κ^{NR} metric seems to provide a clear indication of the compatibility, or lack thereof, in the task set. While useful, it does not tell us why strict compatibility is lost even for case 1.

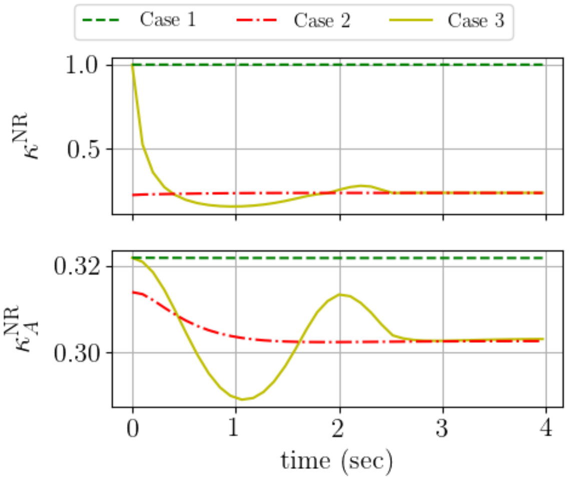


Figure 6.3 – Nuclear norm ratios κ^{NR} and κ_A^{NR} calculated for each case using the standard task-only formulation (see (4.11)) and the augmented equations of motion formulation (see (4.13)), respectively.

Augmented Nuclear Norm Ratio

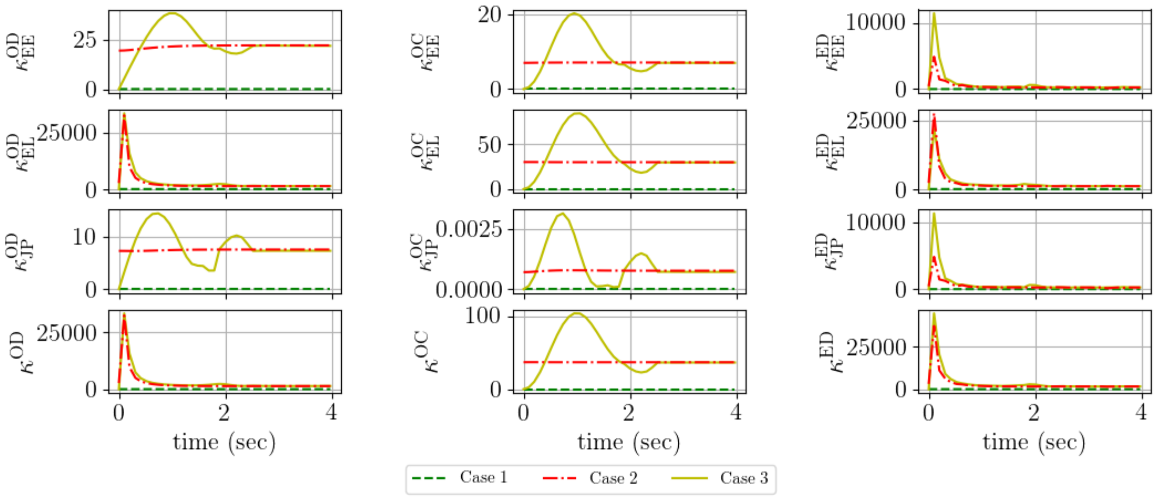
If the tasks in case 1 are perfectly compatible according to κ^{NR}, then strict compatibility should be maintained. To understand this discrepancy, one must understand how compatible the tasks are with the system dynamics. At the bottom of Fig. 6.3, κ_A^{NR} is plotted for each case using the augmented $\tilde{E}_A$ and $\tilde{\mathbf{f}}_A$ formulations from (4.12), which incorporate the equations of motion as well. Looking at these graphs, dramatically different evolutions of the task compatibility across the cases are observed. No longer is case 1 maximally compatible, but rather, only slightly more compatible than case 2. Case 3 fluctuates greatly with the progression of the trajectory and eventually settles near the compatibility measured for case 2. These measures help understand why even seemingly perfectly compatible tasks, as in case 1, may not be compatible with the system dynamics. Because the tasks are not fully compatible with the dynamics, the equality constraints imposed by the equations of motion produce a controller optimum which does not fully satisfy all three task objective functions. Therefore, we can conclude that it is a lack of compatibility between the tasks and the dynamics which causes strict compatibility to be lost and starts the settling behavior.

Optimum Distance

Figure 6.4a shows the Optimum Distance, κ^{OD}, metric for the three cases. The top 3 graphs show the κ_i^{OD} for each task and the bottom graph shows their sum. Looking at case 1, we can see that the metrics measure nearly zero for all three tasks. This makes sense given that the tasks are designed to keep the robot exactly where it starts, and the $\boldsymbol{\chi}_i^*$ needed to do so is roughly the same for all of the tasks. For the elbow task, the $\kappa_{\mathrm{EL}}^{\mathrm{OD}}$ in cases 2 and 3 reach large values which saturate the sum of the metrics, κ^{OD}. This occurs because the elbow task optimum[1], $\boldsymbol{\chi}_{\mathrm{EL}}^*$, distances itself (in a Euclidean sense) from $\boldsymbol{\chi}^*$ in the first time steps for cases 1 and 2. This can be seen in Fig 6.5 where the norm of $\boldsymbol{\chi}_{\mathrm{EL}}^*$ is plotted. This simply indicates that the elbow task optimum has changed but does not prove that the prioritized optimum, $\boldsymbol{\chi}^*$, has not changed. To ensure that the growth of the κ^{OD} metric is indeed caused solely by change in $\boldsymbol{\chi}_{\mathrm{EL}}^*$, the norm of $\boldsymbol{\chi}^*$ is also plotted in Fig. 6.5. Looking at the variations in $\|\boldsymbol{\chi}^*\|$ it is clear that $\boldsymbol{\chi}^*$ moves very little compared to $\boldsymbol{\chi}_{\mathrm{EL}}^*$, and the increase in κ^{OD} is due only to the change in $\boldsymbol{\chi}_{\mathrm{EL}}^*$.

---

[1] See Sec. 4.2.2 for task, prioritized, and controller optimum definitions.



(a) κ^{OD} (b) κ^{OC} (c) κ^{ED}

Figure 6.4 – Computation of the task compatibility metrics for the three compatibility cases studied here. (a) shows the Optimum Distance (OD) metric. (b) shows the Optimum Cost (OC) metric. (c) shows the Ellipsoid Distance (ED) metric.

Optimum Cost

Despite the drastic change in $\boldsymbol{\chi}^*_{\mathrm{EL}}$, what is interesting is that a similar drastic increase is *not* observed in the Optimum Cost metrics, κ^{OC}. In Fig. 6.4b, the cost metrics for each task as well as their sum are plotted for the three cases. Given the evolution of $\kappa^{\mathrm{OD}}_{\mathrm{EL}}$ one would expect to see very large cost increases for the elbow task in cases 2 and 3; however, the rise in $\kappa^{\mathrm{OC}}_{\mathrm{EL}}$ is not comparable with that of $\kappa^{\mathrm{OD}}_{\mathrm{EL}}$. Therefore, even though $\boldsymbol{\chi}^*$ and $\boldsymbol{\chi}^*_{\mathrm{EL}}$ are far in a Euclidean sense, $\boldsymbol{\chi}^*$ does a reasonable job of minimizing $f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EL}}$. To understand why this occurs, $\boldsymbol{\chi}^*_{\mathrm{EL}}$ and $\boldsymbol{\chi}^*$ are mapped into the objective space of $f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EL}}$ and the norm their difference,

$$\|E_{\mathrm{EL}}\boldsymbol{\chi}^*_{\mathrm{EL}} - E_{\mathrm{EL}}\boldsymbol{\chi}^*\| = 5.46, \tag{6.3}$$

shows that the two solutions are not as discordant as would be expected from observing the κ^{OD} metric data. This happens because the null spaces of the task optima are not well captured by distance metrics because defining an optimum inherently requires that all dimensions of $\boldsymbol{\chi}$ be specified. In Euclidean space, these values, which have no impact on the task objective functions, contribute to the norms.

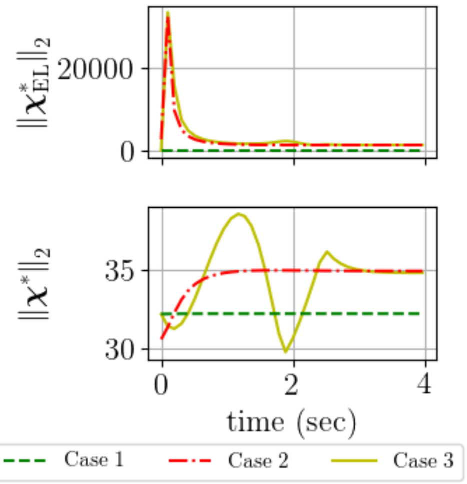


Figure 6.5 – These figures show the variations of the optimal values for the elbow position task, $\boldsymbol{\chi}^*_{\mathrm{EL}}$, and the prioritized optimum, $\boldsymbol{\chi}^*$, for all three cases and demonstrate that the displacement of $\boldsymbol{\chi}^*_{\mathrm{EL}}$ modifies the κ^{OD} metrics shown in Fig. 6.4a, not the changes in $\boldsymbol{\chi}^*$.

Ellipsoid Distance

Given the distance of $\boldsymbol{\chi}^*_{\mathrm{EL}}$ from the other task optima, the compatibility ellipsoid is likewise highly skewed and the Ellipsoid Distance metrics, κ^{ED}, are therefore not very instructive. Looking at Fig. 6.4c, all of the κ^{ED} metric evaluations for the different tasks in cases 2 and 3 show what should be interpreted as incompatibility. However, looking at Fig. 6.6, it is clear that these large metric values are due to the displacement of the compatibility ellipsoid center, $\boldsymbol{c}_\kappa$, with respect to the prioritized optimum, $\boldsymbol{\chi}^*$. The ellipsoid center is displaced as a result of the displacement of $\boldsymbol{\chi}^*_{\mathrm{EL}}$ in cases 2 and 3. Case 1 does not exhibit these behaviors because its elbow task does not change.

Ellipsoid Volume

Unfortunately, by observing the κ^{EV} metrics for each case in Fig. 6.6, it is clear that something is not correct with the compatibility ellipsoid calculations. In theory, the volumes of the compatibility ellipsoids should be greater for the incompatible cases, than that of the compatible case; however, the contrary is presented. This is due to the sparsity of the problem.

In case 1, the three task optima are nearly identical and the compatibility ellipsoid problem is trying to fit a 12-dimensional ellipsoid to 3 closely spaced points. After a few iterations, the SDP solver ascertains that the problem is unbounded and stops. This results in a well centered ellipsoid, but with radii that do not tightly fit the task optima. In cases 2 and 3, the solver never converges on a solution, as a result of the sparsity of the given problem, and returns the last best approximation of the compatibility ellipsoid. The result in this case is a highly skewed (long in one dimension) ellipsoid which, again, poorly fits the task optima. Interestingly, because case 1 is unbounded, the solver returns an ellipsoid which is much larger in volume than the ellipsoids calculated in cases 2 and 3. This occurs because one of the radii of the case 1 compatibility ellipsoid explodes numerically in the unbounded direction. The fluctuations in κ^{EV} for case 1 are the result of numerical instabilities in the SDP problem due to the ill-conditioning of B_κ.

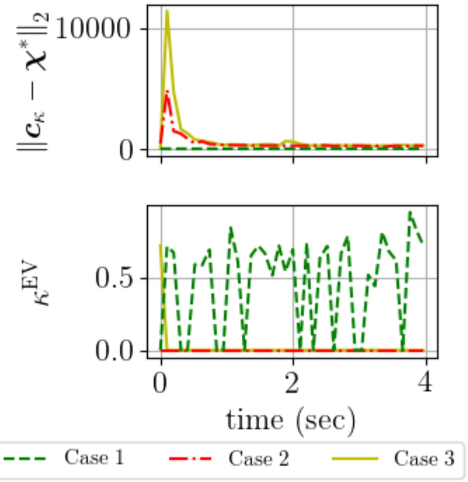


Figure 6.6 – Shows the distance between the compatibility ellipsoid center and the prioritized optimum and the Ellipsoid Volume (EV) compatibility metrics. The computation of the compatibility ellipsoid is highly sensitive to the sparsity of this problem.

6.4 Experimental Setup: Task Feasibility

For these tests, the goal is to understand which task feasibility metrics best identify and quantify the feasibility of a single task. To do so, the same three basic scenarios from the task compatibility study in Sec. 6.2 are used: when the task is completely feasible, completely infeasible, and goes from feasible to infeasible via a trajectory. For the following, only a joint position, or postural, task ($f^{\dot{\boldsymbol{\nu}}}_{\mathrm{JP}}$) is used. A regularization term (see (4.1)) on $\boldsymbol{\chi}$ is also used with $w_0 = 1e-5$. The task and constraint parameters are provided in Tables 6.3 and 6.4.

Table 6.3 – Task set for the task feasibility experiments. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight** w	**Gain** K_p
Joint Position	All DoF	6	1	10

Table 6.4 – Constraints for task feasibility experiments.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	6×2
Torque Limit	All DoF	6×2

To study the effects of task feasibility, three cases are explored. For each of the three cases, the robot starts its motion in the $\boldsymbol{q}_V$ posture and the movement is executed for 5 seconds.

1. **Feasible Task**

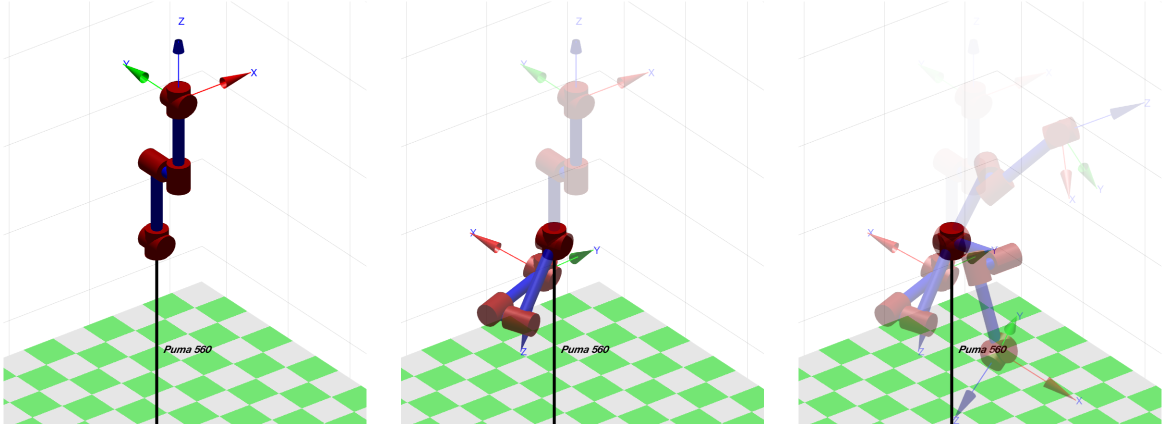
   In this case, the postural task is set to track its initial posture, $\boldsymbol{q}_V$, (see Fig. 6.7a), i.e. not move. Given that the posture held is $\boldsymbol{q}_V$, the robot should be in a state which needs zero torque and is farthest from the joint position limits. This scenario therefore represents a control for the feasibility metrics, and any estimate of feasibility should indicate that it is at a maximum.

2. **Infeasible Task**

   In this case, the postural task reference is set immediately to $\boldsymbol{q}_{IF} = \boldsymbol{q}_{\min} - 10°$. This reference is outside of the joint position range and should therefore violate the joint position limit constraints. In Fig. 6.7b, this is depicted by the overlay of the initial posture, $\boldsymbol{q}_{\mathrm{V}}$, and the final posture where the robot stops as it tries to attain $\boldsymbol{q}_{\mathrm{IF}}$.

3. **Infeasible Task Following a Trajectory**

   In this final case, the same posture reference from case 2 is used but instead of directly setting it as the reference values used in (2.52), it is used to generate a minimum jerk trajectory from the starting $\boldsymbol{q}_V$ posture to the desired infeasible posture, $\boldsymbol{q}_{IF}$, as shown in Fig. 6.7c. Knowing that the final desired posture is incompatible, an evolution in the task feasibility should be observed as the task becomes increasingly infeasible.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 6.7 – The robot motions for the three task feasibility cases. In case 1, the robot maintains its initial feasible posture. In case 2, the reference posture is set immediately to $\boldsymbol{q}_{\mathrm{IF}}$, an infeasible posture, and the robot moves quickly to the reference. In case 3, a joint-space task trajectory is used to move the robot more gradually to $\boldsymbol{q}_{\mathrm{IF}}$.

6.5 Results: Task Feasibility

In this section the feasibility test cases are analyzed using the task feasibility metrics. Using these results, we are able to determine which metrics are useful for whole-body control problems.

6.5.1 Feasibility Metrics

To demonstrate the evolution of the overall movements, the joint torques and positions for each case are presented in Fig. 6.8. In case 1, the robot should not move from $\boldsymbol{q}_V$. This can be confirmed by looking at the joint position evolution and seeing that there are no variations in joint position. Additionally, no torque is needed to maintain $\boldsymbol{q}_V$ and this is clear from the joint torque evolution.

In case 2, the task reference is immediately set to $\boldsymbol{q}_{IF}$ and this causes the robot to move rapidly towards its lower articular limits. Once the limits are hit, as indicated by the position graphs, the robot stops moving. This stop necessitates a sharp peak in torque which can be seen as the spikes on the torque curves. Similar results are found for case 3, but the joint-space task trajectory used to guide the movement causes the robot to reach its limits more slowly. Again, torque spikes at the joint limits can be observed.

Optima Distance

The exploration of the various task feasibility metrics is started by first looking at the Optima Distance (OD) feasibility metric, ϕ^{OD}. Since only one task is being executed, $\phi^{\mathrm{OD}}_{\mathrm{JP}} = \phi^{\mathrm{OD}}$. At the top of Fig. 6.9a, the evolution of ϕ^{OD} is presented. The postural task optimum for case 1 is almost perfectly centered in the feasibility ellipsoid. This makes sense given that the task was designed to maintain the system at a state where $\boldsymbol{\chi}$ is farthest from all limits. This point corresponds roughly to the center of the feasibility ellipsoid. Looking at case 2, because the initial task reference is set to $\boldsymbol{q}_{IF}$ at the beginning of the control horizon, its ϕ^{OD} curve actually starts at a non-zero value. This value continues to grow until it saturates at the moment of hitting the lower joint limits. For case 3, ϕ^{OD} shows similar behavior, but grows more gradually than the ϕ^{OD} of case 2 because of the task trajectory. Once the robot hits the joint limits, ϕ^{OD}

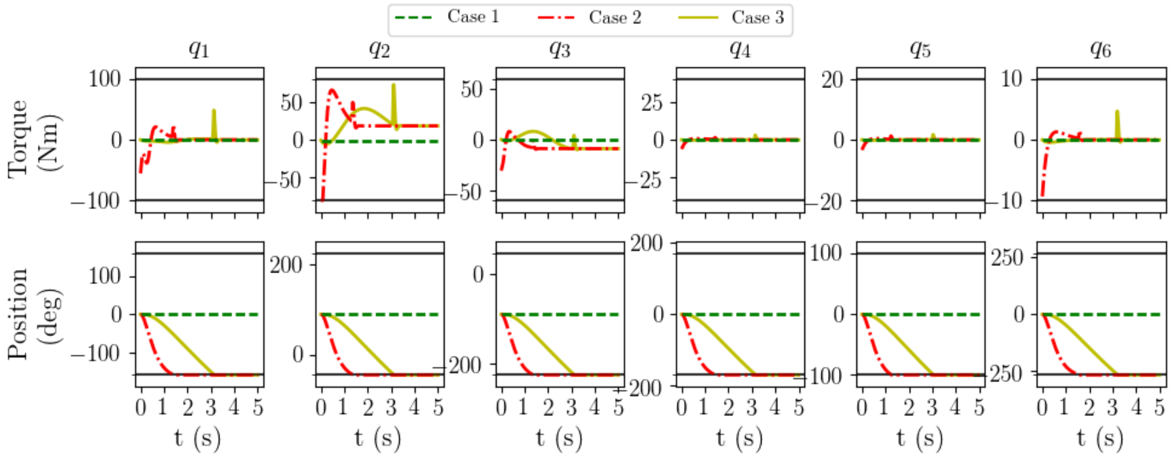
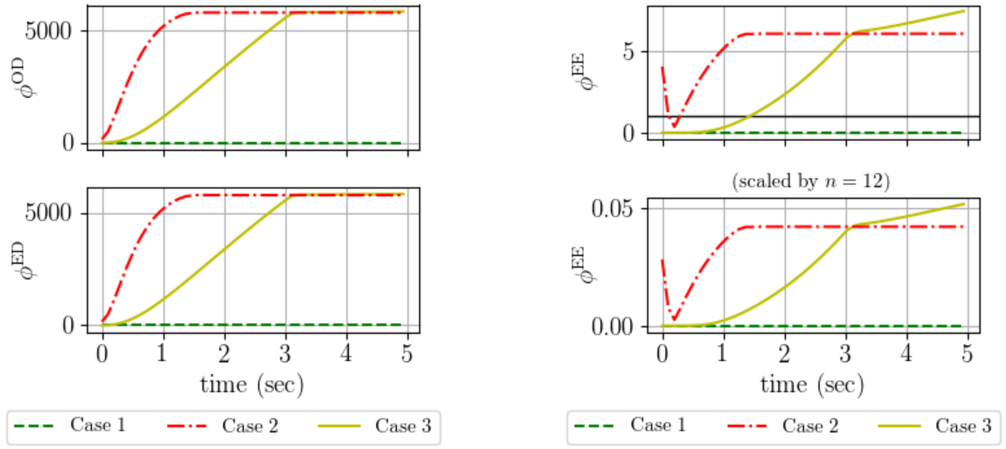


Figure 6.8 – Evolution of the joint torques and positions for each of the three feasibility case studies. The upper and lower bounds are indicated by the darkened black horizontal lines near the y-axis upper and lower limits on each graph.

for case 3 also saturates. The overall behaviors of the curves match what is expected given the design of the tasks, but their values are not easily interpreted in and of themselves.



(a)

(b)

Figure 6.9 – (a) Optima Distance, ϕ^{OD}, and Ellipsoid Distance, ϕ^{ED}, metrics for cases 1-3 of the task feasibility experiment. Because there is only one task in this experiment, $\phi^{\text{OD}} = \phi^{\text{ED}}$. (b) Ellipsoid Evaluation metric, ϕ^{EE}, with and without scaling for cases 1-3 of the task feasibility experiment. The black horizontal line on the upper graph indicates the value of 1.0.

Ellipsoid Distance

Looking now at the Ellipsoid Distance (ED) metrics, ϕ^{ED}, at the bottom of Fig. 6.9a, it can be seen that it is exactly equal to the ϕ^{OD} metrics. This is because there is only one task in the controller and the compatibility ellipsoid is simply a point at $\boldsymbol{\chi}^*_{\mathrm{JP}}$, thus $\phi^{\mathrm{OD}}_{\mathrm{JP}} = \phi^{\mathrm{OD}} = \phi^{\mathrm{ED}}_{\mathrm{JP}} = \phi^{\mathrm{ED}}$.

Ellipsoid Evaluation

The Ellipsoid Evaluation (EE) metrics, ϕ^{EE}, are presented in Fig. 6.9b. In the upper graph, the conservative ϕ^{EE} formulation from (3.70) is shown, and in the lower graph, the less conservative scaled version from (3.72) is shown. Again, because there is only one task, $\phi^{\mathrm{EE}}_{\mathrm{JP}} = \phi^{\mathrm{EE}} = \phi^{\mathrm{EE}^*}$.

For case 1, ϕ^{EE} remains very close to zero, which indicates that it is well inside the feasibility ellipsoid, and near the center. This confirms that the postural task is maximally feasible. For case 2, the value of ϕ^{EE} starts at approximately 4.0, which indicates that the task optimum is likely infeasible. As the robot moves towards $\boldsymbol{q}_{IF}$, the value dips into the feasible $\phi^{\mathrm{EE}} < 1$ region, but then returns to being infeasible, i.e. $\phi^{\mathrm{EE}} > 1$.

This dip can be understood by looking at Fig. 6.10, where the torque lower limit constraint equation for joint 2 is plotted. At the beginning of the movement, there is a large error between the current and desired postures and this generates a $\boldsymbol{\chi}^*_{\mathrm{JP}}$ with a joint torque which exceeds the lower torque limit for joint 2. Where the constraints would be violated because $G\boldsymbol{\chi}^*_{\mathrm{JP}} > \boldsymbol{h}$, the $G\boldsymbol{\chi}^*_{\mathrm{JP}}$ line is darkened. As the robot moves towards the desired infeasible posture, the error is reduced and thus the optimal torque from $\boldsymbol{\chi}^*_{\mathrm{JP}}$ reduces as well. This causes $\boldsymbol{\chi}^*_{\mathrm{JP}}$ to temporarily move towards the feasibility ellipsoid, and thus the short-lived increase in task feasibility.

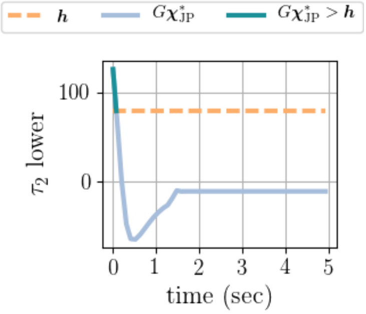


Figure 6.10 – Torque limit lower bound inequality constraint on the q_2 articulation for case 2.

For case 2, as the robot moves towards $\boldsymbol{q}_{IF}$, the ϕ^{EE} values quickly increase again and saturate at the joint limits. When the robot hits its limits the task reference values stop changing, and this causes the saturation. In case 3, we observe a similar trend towards infeasibility, which comes as no surprise, but this time the ϕ^{EE} values begin near zero as with case 1. The ϕ^{EE} curve for case 3 does not saturate like case 2, because the joint-space task trajectory continues to change the task reference values, despite the robot being stopped by its joint limits.

Scaled Ellipsoid Evaluation

At the bottom of Fig. 6.9b, the scaled feasibility ellipsoid is used to calculate ϕ^{EE}. The overall behaviors of the curves remain unchanged; however, their ϕ^{EE} values are smaller and indicate that the tasks remain feasible for all cases. This is known to be false and stems from the fact that the ellipsoid is scaled with the dimension, n, of the problem in order to ensure that it encloses the polyhedron defined by the inequality constraints. Accordingly, the ellipsoid overly approximates the feasible region of solutions and wrongly indicates that the task optima are feasible. To confirm that the task becomes infeasible, the 6 inequality constraints, which govern the lower joint position limits, are evaluated at the prioritized optimum and plotted for case 3 in Fig. 6.11. This shows that the task is infeasible and these constraints are violated by the task optimum, i.e. in the active set. Knowing that the task is infeasible, yet the scaled ϕ^{EE} metric indicates otherwise, this metric may be unsuited for humanoids where n can be much larger.

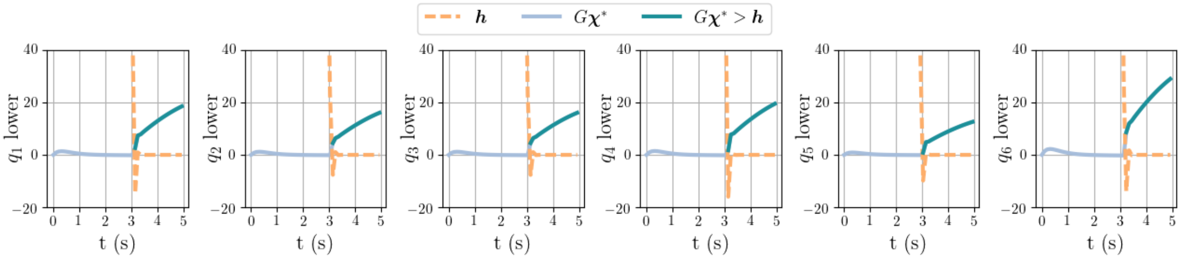


Figure 6.11 – The evolution of the inequality constraints for the lower joint position limits for case 3, evaluated at the prioritized optimum, $\boldsymbol{\chi}^*$. The $G\boldsymbol{\chi}^*$ line is darkened where the prioritized optimum would violate the constraint. A violation indicates that the prioritized optimum is infeasible and thus activates certain constraints in the QP.

Ellipsoid Volume

Finally looking at the ϕ^{EV} metric in Fig. 6.12, one can observe an interesting phenomenon: for all three cases, the volumes of the feasibility ellipsoid remain constant. The values of ϕ^{EV} are expectedly large; however, being in the range of a 300 octillions[2], may cause reason for doubting the utility of such a measure. Nevertheless, the problem explored here is not highly constrained and may not be the best case study for this particular metric.

In the middle of Fig. 6.12, the norm of the feasibility ellipsoid center for each case are plotted. At the bottom of Fig. 6.12, the norm of the $\boldsymbol{\chi}^*_{\text{JP}}$ for each case are plotted. Looking at these two plots for cases 2 and 3 reveals that the ellipsoid centers are moving great distances, while the task optima are moving much less comparatively. Therefore, more so than measure how far the task optima distance themselves from the constraints, the ellipsoid based metrics are measuring how far the constraints are moving from the optima. While purely semantic in nature, these metrics show that the infeasibility has more to do with the variation in the inequality constraint matrix, G, than in the task objective functions. That being said, the constraint equations depend on the dynamics of the robot, which are governed by the physics of the motion produced by the tasks and the environmental interactions. Finally, given that the ellipsoid volume does not change (its radii remain constant) but the center moves, we can observe that the feasibility ellipsoid translates in the space of $\boldsymbol{\chi}$ as the constraints change.

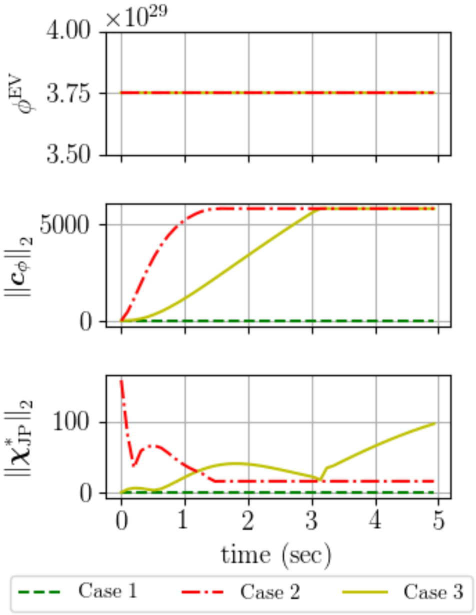


Figure 6.12 – (top) the Ellipsoid Volume metric, ϕ^{EV} for the task feasibility experiment. (middle) the norm of the feasibility ellipsoid center. (bottom) the norm of the postural task optimum.

---

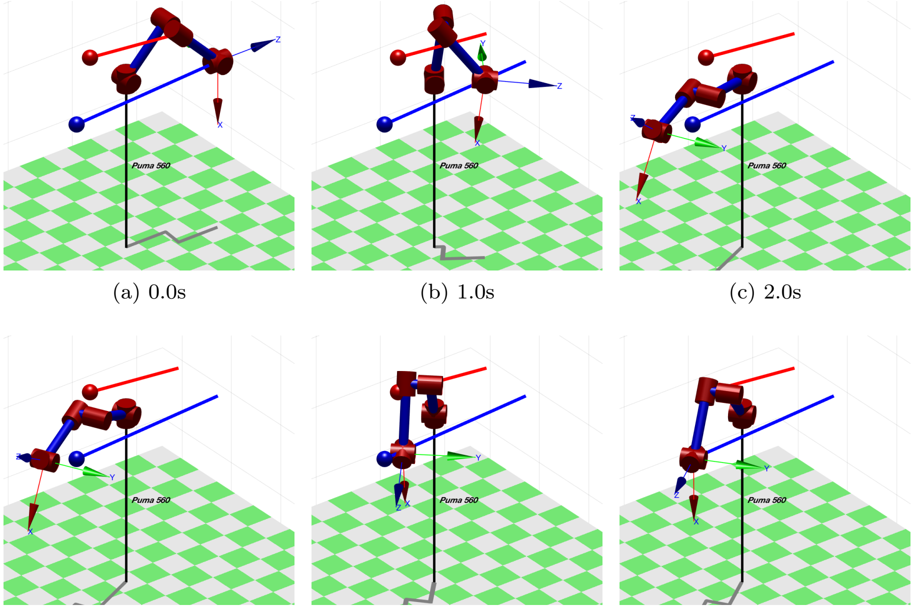
[2] One octillion = 1×10^{27}

6.6 Experimental Setup: Temporal Task Compatibility and Feasibility

In this experiment, a more realistic task compatibility and feasibility scenario is investigated. Using the same tasks from Sec. 6.2, as shown in Table 6.2 and the constraints given in Table 6.5, a single example is explored where the elbow and EE tasks are guided by minimum jerk trajectories to two kinematically feasible and compatible goal positions. Figure 6.13 shows a time-lapse of the resulting motion. These trajectories follow straight paths from the starting task positions to the goal positions, and during portions of the trajectory, are kinematically incompatible. The postural task maintains its original reference throughout the motion. Further complicating the scenario, the tasks should drive some of the articulations to their position limits. This set of tasks and trajectories should then start as being both compatible and feasible, become incompatible and infeasible, and finish by being both compatible and feasible again.

Table 6.5 – Constraints for the temporal task compatibility and feasibility experiments.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	6×2
Torque Limit	All DoF	6×2



(a) 0.0s (b) 1.0s (c) 2.0s

(d) 4.0s (e) 6.0s (f) 8.0s

Figure 6.13 – Time-lapse of the motion used to explore the temporality of task compatibility and feasibility.

6.7 Results: Temporal Task Compatibility and Feasibility

An approximate understanding of the compatibility of the tasks can be observed by looking at their real and reference position values throughout the movement in Fig. 6.14. From these graphs, it can be seen that the poor trajectory tracking at the beginning of the motion causes a large overshoot at around 2 seconds. This can also be seen in Fig. 6.13c. If the tasks were perfectly compatible/feasible, then there would be no tracking errors, because each task objective function would be perfectly optimized. The overshoot then indicates that the task compatibility and/or feasibility is reduced in the middle of the motion and then increases at the end. This is by design of course.

To determine whether the overshoot is the result of task incompatibility and/or task infeasibility, the joint position and torque evolutions are plotted in Fig. 6.15. Here, the joint torques remain well within their bounds; however, the joint positions hit their lower limits. Specifically, q_1 hits its lower limit between 2 and 4 seconds into the motion, while q_3 hits its lower limit only briefly between 1 and 1.5 seconds. The fact that the joints hit their constraints indicates that there is assuredly an infeasibility in the task set between the moments of 2 and 4 seconds. However, this does not negate the existence of incompatibilities between the tasks.

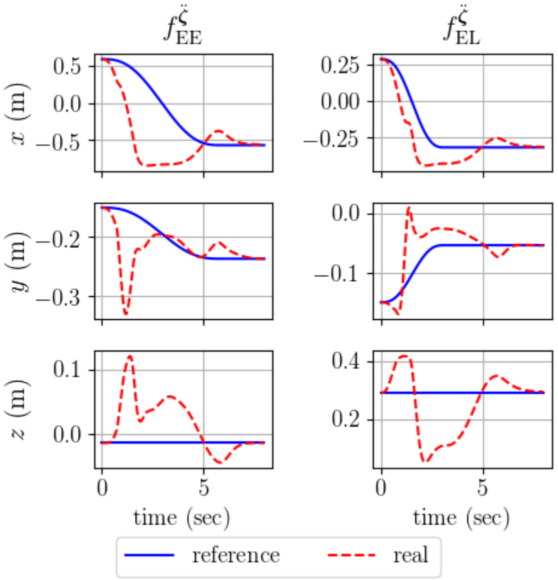


Figure 6.14 – Task reference and real position values throughout the movement. As can be seen, the trajectory tracking suffers in the middle of the movement, which indicates some decrease in task compatibility/feasibility.

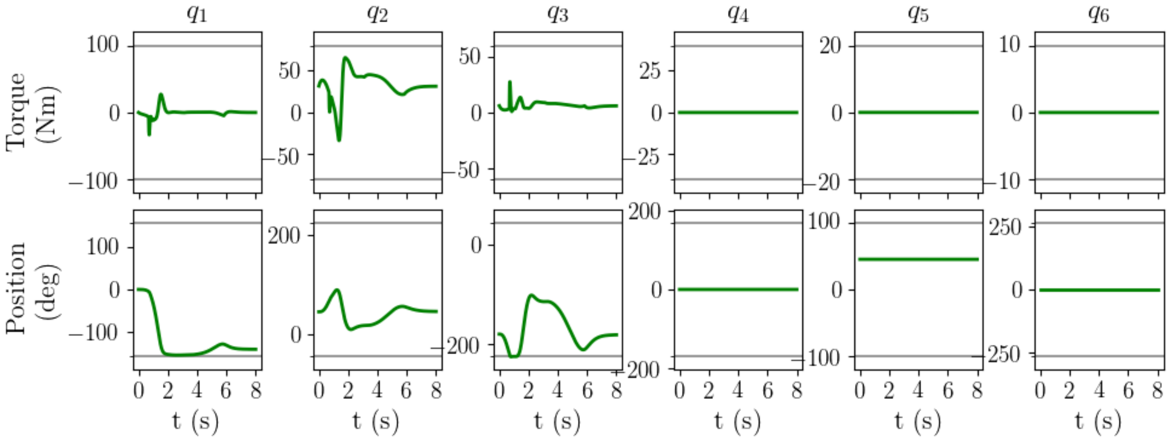


Figure 6.15 – The evolution of the joint torques and positions for the temporal example. During the movement, q_1 and q_3 hit their lower position limits, thus indicating an infeasibility.

6.7.1 Compatibility Metrics

To begin the analysis of the example problem, first the compatibility metrics are computed. For the sake of clarity, the ellipsoid based metrics which are found to be numerically sensitive in the task compatibility experiment (see Sec. 6.2), are omitted from the analyses.

Nuclear Norm Ratio

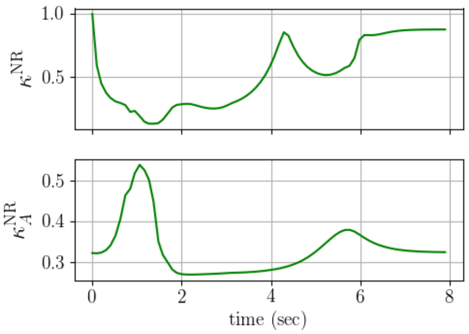


Figure 6.16 – Top: κ^{NR} calculated using the standard task-only formulation (see (4.11)). Bottom: κ_A^{NR} calculated using the augmented equations of motion formulation (see (4.12)).

In Fig. 6.16 the Nuclear norm Ratio, both standard and augmented, are presented. Looking first at the standard formulation of κ^{NR} at the top of Fig. 6.16, the curve indicates exactly what is expected from the movement, based on the trajectory design. Initially, the tasks start out as completely compatible with $\kappa^{\mathrm{NR}} = 1$ and in the middle of the movement the compatibility decreases. This can be seen to correspond to the poor trajectory tracking presented in Fig. 6.14. Interestingly, between 3.0 and 4.5 seconds, the compatibility increases, and then from 4.5 to 5 seconds decreases again. These periods correspond to the first overshoot recovery. Accumulated tracking errors generate high desired accelerations, which cause the robot to aggressively servo the EE and elbow goal positions, resulting in a second overshoot (see Fig. 6.13e), and then finally convergence. The κ^{NR} values do not return to 1 in this case because the postural task references remain at $\boldsymbol{q}_N$ and to satisfy the more heavily weighted EE and elbow tasks, the posture must be different from $\boldsymbol{q}_N$, thus inducing an incompatibility with the postural task.

Augmented Nuclear Norm Ratio

At the bottom of Fig. 6.16, the augmented κ_A^{NR} metric is presented, and paints a different picture from the one presented using the standard metric. Here, the compatibility starts relatively low, peaks at around 1 second and then again between 5 and 6 seconds. These peaks in κ_A^{NR} correspond to the accelerations during the overshoots. Looking at the desired task acceleration norms for the three tasks in Fig. 6.17, one cannot see any clear correlation between the magnitude of the desired acceleration vectors of the tasks and the phases of compatibility and incompatibility according to the κ_A^{NR} values. This suggests that factors, other than those measured by the metrics here, may have an impact on the compatibility of the tasks with the dynamics.

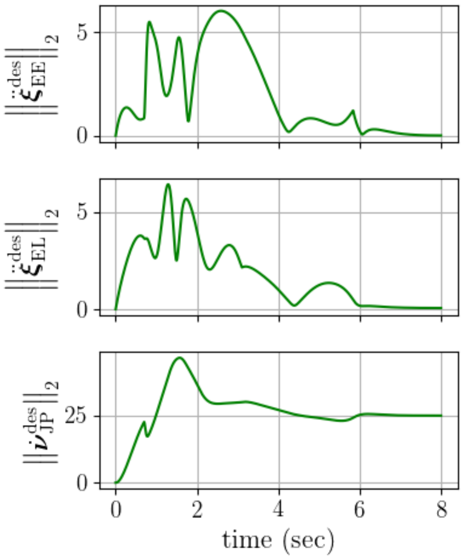
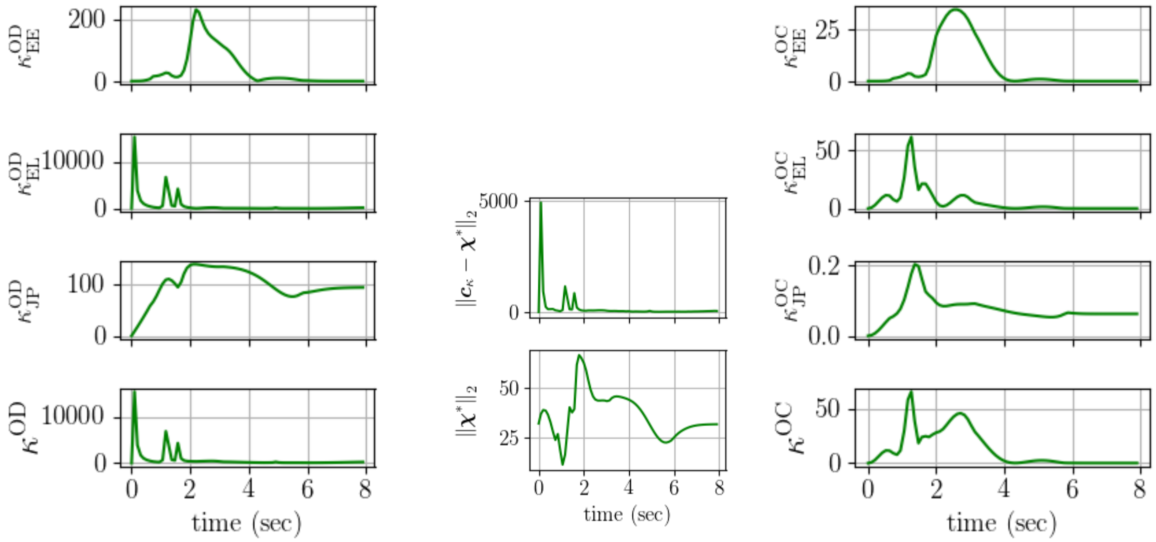


Figure 6.17 – The desired acceleration norms for each task.

Optimum Distance

Looking now at the κ^{OD} metrics in Fig. 6.18a, we see the same displacement of the elbow task optimum as in the incompatible cases studied in Sec. 6.2. The values of $\kappa_{\mathrm{EE}}^{\mathrm{OD}}$ and $\kappa_{\mathrm{JP}}^{\mathrm{OD}}$ reflect the expected evolution of their compatibility with the prioritized optimum — i.e. both tasks are initially compatible and become incompatible in the middle of the motion. The EE task, however, returns to being highly compatible with $\boldsymbol{\chi}^*$, while the postural task remains largely incompatible with $\boldsymbol{\chi}^*$.

In Fig. 6.18b, the Euclidean distance between $\boldsymbol{\chi}^*$ and the ellipsoid center, $\boldsymbol{c}_\kappa$, and the norm of $\boldsymbol{\chi}^*$ are plotted. Here, it can be seen that $\boldsymbol{\chi}^*_{\text{EL}}$ changes greatly, and while there is some fluctuation in $\boldsymbol{\chi}^*$, it is orders of magnitude smaller than that of $\boldsymbol{\chi}^*_{\text{EL}}$. This again results in a skewing of the compatibility ellipsoid and a displacement of its center. Given the poorly conditioned nature of the compatibility ellipsoid, further ellipsoid based compatibility metrics are ignored here because the analyses are identical to those from Sec. 6.2.



(a) (b) (c)

Figure 6.18 – (a) Optimum Distance metrics, κ^{OD}, for the temporal example. (b) (top) the Euclidean distance between $\boldsymbol{\chi}^*$ and the ellipsoid center, $\boldsymbol{c}_\kappa$, and (bottom) the norm of $\boldsymbol{\chi}^*$. (c) Optimum Cost metrics, κ^{OC}.

Optimum Cost

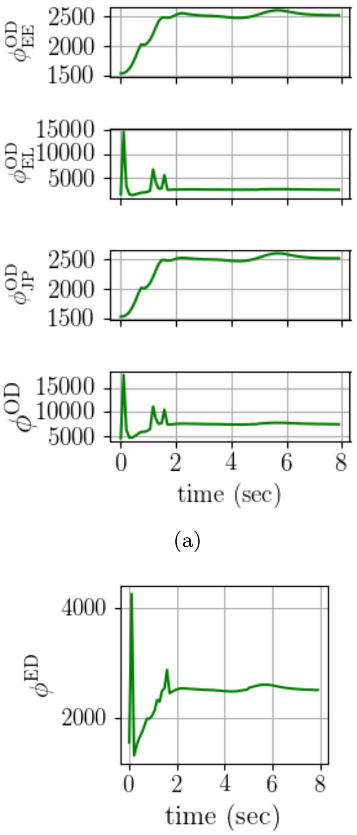
Ignoring the ellipsoid based metrics leaves the κ^{OC} compatibility metrics, which are plotted in Fig. 6.18c. Looking at the curves for $\kappa^{\text{OC}}_{\text{EE}}$, $\kappa^{\text{OC}}_{\text{EL}}$, and $\kappa^{\text{OC}}_{\text{JP}}$, it is easy to interpret the evolution of the individual task compatibilities with the ensemble. This is synthesized nicely in the sum of their measures, κ^{OC}. The motion starts by being compatible, becomes incompatible, and ends by being compatible. A pleasant artifact of summing the κ^{OC} task metrics is that the weighting of the postural task comes into account, and even though it is mostly incompatible with the prioritized optimum, it does not greatly affect the global compatibility of the task set.

6.7.2 Feasibility Metrics

With the task compatibility metrics analyzed, we move on to the task feasibility metrics.

Optima Distance

Similarly to the κ^{OD} metrics, the elbow task dominates the ϕ^{OD} metric, because it distances itself greatly from its initial optimum at the start of the movement. In Fig. 6.19a, the component ϕ^{OD} values are plotted for each task, along with their sum. Because of the magnitude of $\phi^{\mathrm{OD}}_{\mathrm{EL}}$, most of the information from the other two tasks, in the middle of the motion, is lost in the sum ϕ^{OD} metric. However, near the end of the motion their contribution to ϕ^{OD} becomes more prominent. Globally speaking, $\phi^{\mathrm{OD}}_{\mathrm{EE}}$, $\phi^{\mathrm{OD}}_{\mathrm{EL}}$, and $\phi^{\mathrm{OD}}_{\mathrm{JP}}$ all tend to values which indicate increasing infeasibility as they get further from the feasibility ellipsoid center over the course of the movement. This goes contrary to what is expected given that the robot does attain its desired goal positions.



(a)

(b)

Figure 6.19 – (a) Optima Distance metric, ϕ^{OD}, and (b) Ellipsoid Distance metric, ϕ^{ED}, for the temporal experiment.

Ellipsoid Distance

Looking at the ellipsoid center distances, ϕ^{ED} indicates the same progression, but with less noise from the elbow task. The Fig. 6.19b, shows a rise in infeasibility over the course of the movement with a few intermittent peaks, which correspond to the rapidly changing elbow task optima.

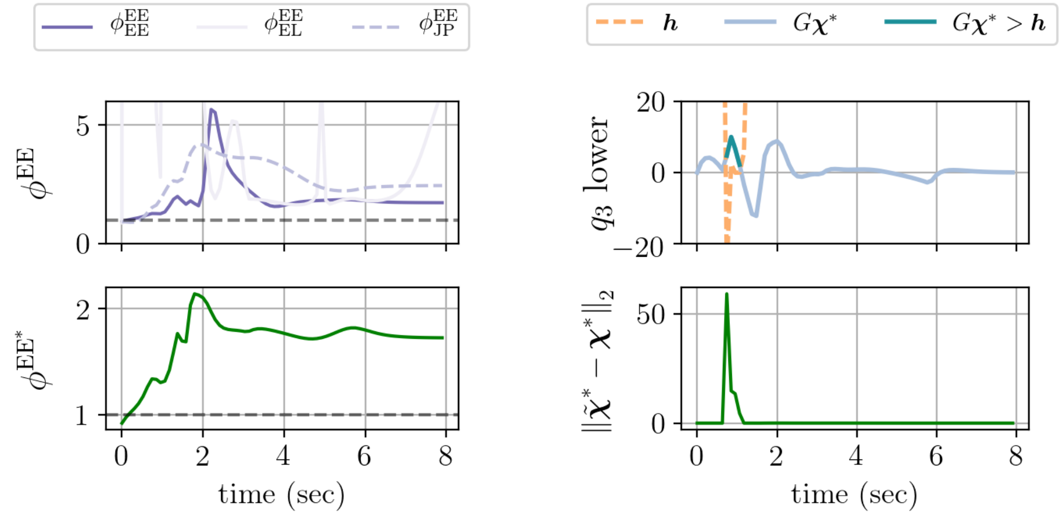
Ellipsoid Evaluation

The same behavior can be observed in the ϕ^{EE} and ϕ^{EE^*} metrics in Fig. 6.20a. The chart of ϕ^{EE} values at the top of Fig. 6.20a has been zoomed in to ignore the outlying $\phi^{\mathrm{EE}}_{\mathrm{EL}}$ values.

All of the ϕ^{EE} metrics indicate that the tasks become infeasible at the end of the motion, counter to what is expected. Looking at the lower position bounds, plotted at the top of Fig. 6.20b, the only moment where the prioritized optimum violates a constraint is at approximately 0.6 seconds on joint 3 (q_3). At the end of the motion, all of the constraints are respected.

At the bottom of Fig. 6.20b, the distance between the prioritized optimum, $\boldsymbol{\chi}^*$, and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$ is plotted. Zero distance indicates that the prioritized optimum is completely feasible with the given constraints. Any positive distance indicates that the prioritized optimum lies outside of the feasible set and cannot be perfectly attained. This only occurs at approximately 0.6 seconds, which corroborates the information gleaned from graph of the q_3 lower bounds inequality. For the rest of the motion, there is zero distance between the controller and prioritized optima.

This indicates that the prioritized optimum is feasible during these control instances. Therefore, it is clear that the feasibility ellipsoid poorly approximates the constraint polyhedron near the region of $\boldsymbol{\chi}^*$, and that this region must be near a vertex. This is supported by the fact that the robot is close to its first and third joint position limits at the end of the motion as shown in Fig. 6.15.



(a)

(b)

Figure 6.20 – (a) shows the Ellipsoid Evaluation metrics for each of the tasks (top) and for the prioritized optimum, $\boldsymbol{\chi}^*$, (bottom). The dashed horizontal black line indicates a value of 1.0. (b) shows the evolution of the inequality constraints for the lower joint position limit of articulation q_3 at $\boldsymbol{\chi}^*$ (top), and the distance between the prioritized optimum and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$ (bottom). Zero distance indicates that the prioritized optimum is completely feasible with the given constraints. Any positive distance indicates that the prioritized optimum lies outside of the feasible set and cannot be perfectly attained.

6.8 Conclusions

From the various compatibility and feasibility case studies in this chapter, we can draw some conclusions about the utility of the different metrics and measures.

6.8.1 Compatibility Metrics

In terms of task compatibility, the first conclusion to be drawn is that the compatibility ellipsoid may be numerically ill-conditioned because the problem is unbounded. In practical terms, since there are so few optima and so many dimensions, even for this 6-DoF robot, it is unlikely that an SDP algorithm will find an ellipsoid which accurately approximates the set. For this reason, the ellipsoid based metrics, κ^{ED} and κ^{EV}, cannot be reliably used for robotic control problems.

Given the large variations of the elbow task optimum (one of the factors in the ill-conditioning of the objective ellipsoid) the κ^{OD} metric must also be used carefully. If only the sum of the κ_i^{OD} measures is looked at, and one of the task κ_i^{OD} is orders of magnitude larger than the others, then their information is lost in summing with the former. The individual κ_i^{OD} do provide some helpful insight as to how the task compatibility evolves, but are mostly qualitative.

The κ^{NR} and κ_A^{NR} metrics, prove highly useful in both qualifying and quantifying the task compatibility. Using the standard κ^{NR} formulation, the overall compatibility of the task sets in the examples is easily interpreted. However, the augmented formula shows that despite being compatible with one another, the tasks are generally incompatible with the system dynamics. This discrepancy between κ^{NR} and κ_A^{NR} merits a more in depth analysis, but suffice it to say that the dynamic compatibility of the tasks is more complex an issue than anticipated.

The κ^{OC} metric also nicely summarizes the overall task compatibility through the cost associated with the movement. By looking directly at the objective function outputs, one can derive quite easily, how well or poorly the tasks are being optimized and this translates itself into good or bad task reference tracking.

6.8.2 Feasibility Metrics

For the same issues of numerical sensitivity, certain task feasibility metrics are shown to be of little qualitative or quantitative value. The elbow task, in the applicable studies, tends to saturate the ϕ^{OD} metric and cover up any useful data from the other task metrics. The same is true for the ϕ^{ED} metric since the elbow task causes a displacement of the compatibility ellipsoid center and therefore saturates this metric as well. Likewise, the ϕ^{EV} metric is shown to be useless in evaluating task feasibility.

For the ϕ^{EE} metrics, since each task is looked at individually, one can derive some useful analyses if the range of values observed stays around 1.0, and the outliers are ignored. The ϕ^{EE^*} synthesizes the feasibility information and filters out the outlying noise in a compact measure of overall task feasibility. While shown to be overly conservative, ϕ^{EE^*} gives a good idea, quantitatively, of how feasible the task set is. Qualitatively, it provides a useful description of the evolution of the task feasibility, and even if it is conservative, it does indicate when constraints are being approached. This can also be confirmed by looking at the constraint equation evolutions, as there are only 24 for this problem. For high dimensional problems (e.g. humanoids), however, analyzing the constraint equations directly is cumbersome and ϕ^{EE^*} is a compact alternative. Measuring the distance between $\boldsymbol{\chi}^*$ and $\tilde{\boldsymbol{\chi}}^*$, is also helpful in determining if constraints are activated by the prioritized optimum, but does not show the evolution of the task feasibility.

6.8.3 Accounting for Null Spaces

From these results, it is clear that ignoring the null spaces of the task objective functions in the calculation of the distance metrics is a poor choice. Tasks, like the elbow task, can move to solution spaces far away from the other tasks because of their null spaces and the minimum norm characteristics of the least squares solution. To account for this, only the column space projection of the vectors should be compared in the distance calculations. For example, for ϕ_i^{OD}, the following could be used,

$$\phi_i^{\mathrm{OD}} = \left\| P_i^C \boldsymbol{c}_\phi - \boldsymbol{\chi}_i^* \right\|_2 \tag{6.4}$$

$$\text{where} \quad P_i^C = E_i (E_i^\top E_i)^{-1} E_i^\top . \tag{6.5}$$

Here, P_i^C is the column space projector of the i^{th} task. While this is a promising solution for improving certain metrics, it is left for future work.

6.8.4 Most Useful Metrics

In summary, the task compatibility metrics to be retained from these experiments are, the Nuclear norm and Augmented Nuclear norm Ratios, κ^{NR} and κ_A^{NR}, as well as the Optimum Cost metric, κ^{OC}, and its components κ_i^{OC}. The task feasibility metrics to be retained are, the Ellipsoid Evaluation metrics, ϕ^{EE} and ϕ^{EE^*}. Using these measures any task and constraint set can be effectively analyzed for compatibility and feasibility. With these tools, we now determine if the model-free TCFM method described in Chapter 5 is capable of improving task compatibility and feasibility without ever directly measuring these phenomena.

Chapter 7

Proof of TCFM Concept

The goal of these experiments is to show that model-free TCFM can truly maximize task compatibility and feasibility. To do so, the example control problem from Chapter 6 is reformulated in the model-free fashion. Comparison of the model-free costs and the model-based metrics shows that the costs do indeed capture the effects of task incompatibility and infeasibility. From here, the TCFM algorithm is used to optimize the task trajectories. Analysis of the optimized motion using the model-based metrics shows that TCFM can maximize task compatibility and feasibility and produce motions which better match their expected behavior.

7.1 Experimental Setup: Model-Free Costs vs. Model-Based Metrics

In this first experiment, the goal is to compare the model-free costs, developed in Chapter 5, with the model-based metrics from Chapter 4. This is done to validate that the model-free cost functions do indeed capture the side-effects of task incompatibility and infeasibility.

In this example, the robot being used is once again the PUMA 560, and the details of the simulation can be obtained from Sec. 6.6. No interactions are possible between the robot and its surroundings, save for the simulated gravitational forces. As a reminder the task set consists of an end-effector (EE) position task ($f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EE}}$), an elbow position task ($f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EL}}$), and a joint position, or postural, task ($f^{\dot{\boldsymbol{\nu}}}_{\mathrm{JP}}$). The task and constraint parameters are provided in Tables 7.1 and 7.2.

Table 7.1 – Task set. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Cartesian	EE	3	1.0	10
Cartesian	Elbow	3	1.0	10
Joint Position	All DoF	6	0.0001	10

Table 7.2 – Constraints.

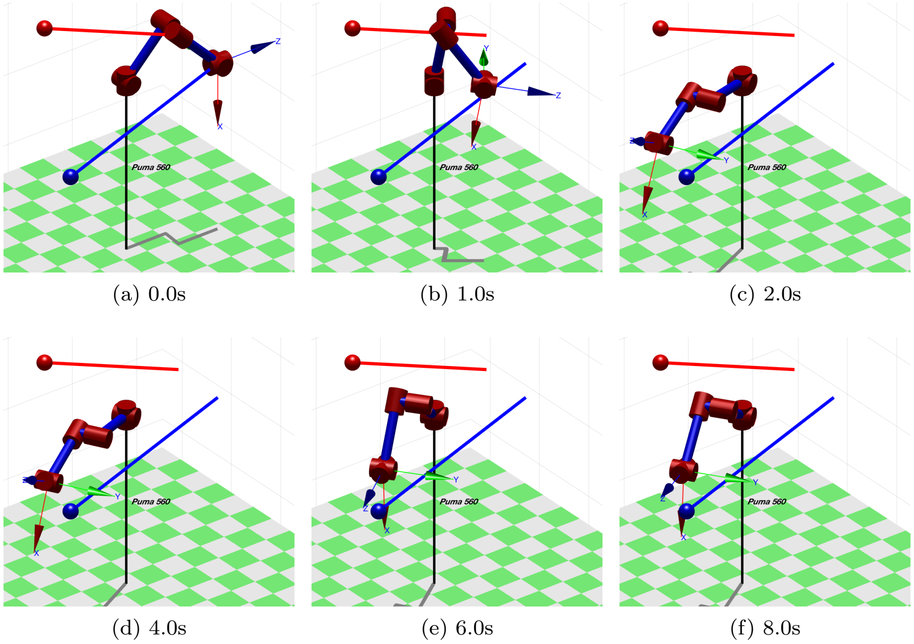
Constraint Type	DoF/Link	Dimension
Joint Limit	All DoF	6×2
Torque Limit	All DoF	6×2

Again, the elbow and EE tasks are guided by minimum jerk trajectories to two goal positions, while the postural task maintains its original reference throughout the motion. The trajectory generators in this example are then based on analytical minimum jerk parameterizations for the elbow and EE tasks, and a set point for the postural task. Each of these trajectory generators takes the initial position of the task and the goal position of the task as input parameters. For the minimum jerk trajectories, the maximum velocity of the trajectory must also be specified, and in this example, it is fixed at $0.2ms^{-1}$.

Two cases are explored to evaluate the cost functions presented in Sec. 5.6. For each case, the goal positions of the tasks, which are the policy parameters, $\boldsymbol{\theta}$, dictate the resulting task trajectories and consequently the policy, $\boldsymbol{\pi}_{\boldsymbol{\theta}}(k)$. Therefore, by changing the goal positions, the policy is changed.

In the first case, the same goal positions used from the original example in Sec. 6.6 are used. The EE and elbow goal positions are kinematically compatible; however, the straight paths of the trajectory, are kinematically incompatible during portions of the movement. This set of tasks and trajectories should then start as being both compatible and feasible, become incompatible and infeasible, and finish by being both compatible and feasible again. Figure 6.13 in Chapter 6 shows a time-lapse of the resulting motion for case 1.

In the second case, the goal positions for the elbow and EE tasks from case 1 are translated to two kinematically incompatible positions. This set of tasks and trajectories should start as being both compatible and feasible and then become increasingly incompatible and infeasible. Figure 7.1 shows a time-lapse of the resulting motion for case 2.



(a) 0.0s (b) 1.0s (c) 2.0s

(d) 4.0s (e) 6.0s (f) 8.0s

Figure 7.1 – Time-lapse of the motion produced by the case 2 policy.

For both cases, the policy duration, $t_{\boldsymbol{\pi}}$, is set to 8.0 seconds to ensure that the primary EE and elbow tasks have sufficient time to complete. The trajectory durations for each task are presented in Table 7.3. Using these values, the policies are rolled out and their performances are evaluated.

case	$t^{\mathrm{EE}}_{n_\lambda}$	$t^{\mathrm{EL}}_{n_\lambda}$	$t^{\mathrm{JP}}_{n_\lambda}$
1	5.8	3.1	8
2	7.4	4.4	8

Table 7.3 – Task trajectory durations in seconds for cases 1 and 2.

7.2 Results: Model-Free Costs vs. Model-Based Metrics

Before evaluating the costs, we first analyze the policy rollouts by looking at the trajectory tracking performance for both cases in Fig. 7.2. Here, the reference, denoted "ref", and real, denoted "real", positions of the EE and elbow tasks are plotted for both case 1 and 2. The case 1 trajectories are in green and the case 2 trajectories are in dashed red. The reference positions are indicated by the lighter lines and the real positions by the darker lines.

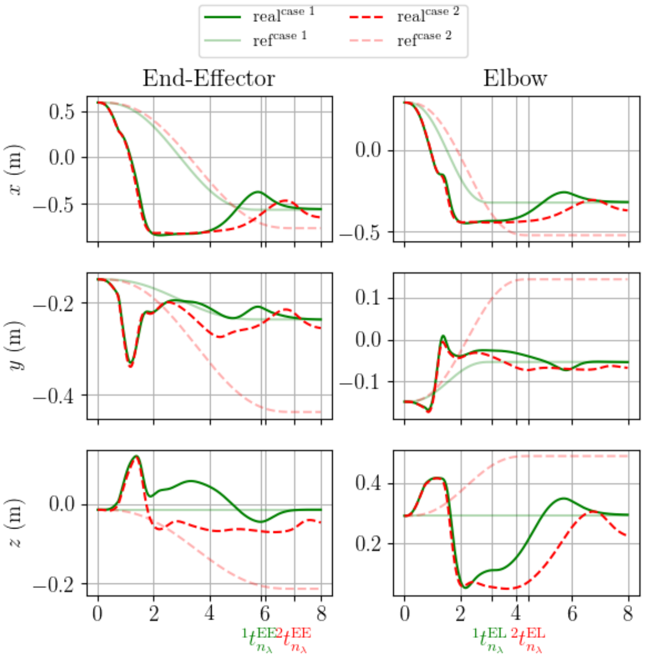


Figure 7.2 – Real and reference values for the EE and elbow task trajectories for the case 1 and case 2 rollouts. The case 1 trajectories are in green and the case 2 trajectories are in dashed red. The reference positions are indicated by the lighter lines and the real positions by the darker lines.

Because case 1 is the same as the scenario from Sec. 6.6, the analyses on the evolution of the two tasks over the rollout remain the same. To briefly reiterate, at the beginning of the movement, the tasks poorly track their references, which are incompatible, and this causes the robot to overshoot its goal position. The robot then oscillates to the goal positions of the EE and elbow task, where tracking error is minimal, but well after the t_{n_λ} for both tasks, which are indicated on the x-axes. The case 1 policy parameters produce tasks which start compatible and feasible, become incompatible and infeasible and end compatible and feasible. This means that the performance cost, j^{P}, for case 1 should go from low to high to low.

For case 2, the initial behavior of the tasks is quite similar to that of case 1 where the tasks overshoot the goal positions in the early phase of the movement due to error build up and the joint limit infeasibilities (see Fig. 6.15). However, the end of the rollout shows that the trajectories never attain their goal positions, which are incompatible by design. Based on this behavior, one would expect a higher performance cost for case 2. Furthermore, the evolution of the performance cost for case 2 should reflect the increasing infeasibility and incompatibility generated by the policy.

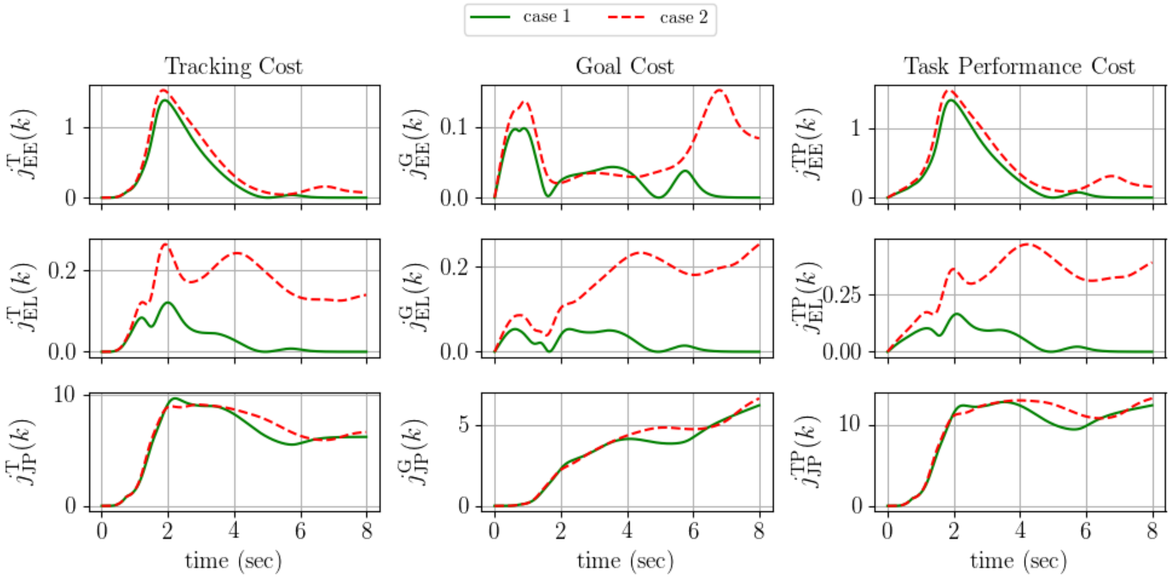


Figure 7.3 – Time evolution of the tracking, $j_i^{\mathrm{TP}}(k)$, goal, $j_i^{\mathrm{TP}}(k)$, and task performance, $j_i^{\mathrm{TP}}(k)$, costs for each task in the example problem from Sec. 6.6. The costs are calculated as the area under the curves, but looking at their evolution allows the task performance to be analyzed.

7.2.1 Model-Free Cost Analysis

To confirm these expected performances, j^{P} must be calculated for each case, and in order to do that, the individual task performance costs, j_i^{TP}, from (5.22), must be evaluated. To calculate each j_i^{TP} the tracking, j_i^{T}, and goal, j_i^{G}, costs for each task are first calculated.

	case 1			case 2		
task	j_i^{T}	j_i^{G}	j_i^{TP}	j_i^{T}	j_i^{G}	j_i^{TP}
$f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EE}}$	239.91	23.09	263.00	322.03	53.31	375.34
$f^{\ddot{\boldsymbol{\xi}}}_{\mathrm{EL}}$	23.72	17.67	41.39	124.14	126.80	250.94
$f^{\dot{\boldsymbol{\nu}}}_{\mathrm{JP}}$	4887.06	2646.09	7533.15	5147.61	2815.04	7962.65

Table 7.4 – The sums of the tracking, goal and task performance costs for cases 1 and 2.

Tracking and Goal Costs In Fig. 7.3, $j_i^{\mathrm{T}}(k)$, $j_i^{\mathrm{G}}(k)$, and $j_i^{\mathrm{TP}}(k)$ for the end-effector (EE), elbow (EL), and joint position (JP), tasks are plotted. Case 1 results are indicated by the solid green lines, and case 2 results, by the dashed red lines. Here, the costs are plotted as calculated at each time step, i.e.,

$$j_i^{\mathrm{T}}(k) = \|\boldsymbol{\epsilon}(k)\|_2^2, \quad j_i^{\mathrm{G}}(k) = \frac{kh}{t_{n_\lambda}}\|\boldsymbol{\epsilon}^{\mathrm{goal}}(k)\|_2^2, \quad \text{and} \quad j_i^{\mathrm{TP}}(k) = j_i^{\mathrm{T}}(k) + j_i^{\mathrm{G}}(k). \tag{7.1}$$

Their sum is then the area under the curves, and these sums are provided by Table 7.4. Looking at their time evolution allows the task performance to be analyzed, and consequently the analysis of the effects of task incompatibility and infeasibility.

Looking at $j_i^{\mathrm{T}}(k)$ and $j_i^{\mathrm{G}}(k)$ for case 1, the EE and EL costs indicate that the tasks are performed well initially, poorly during the first 5 seconds, then well at the end of the rollout. For the JP task, however, the performance continues to degrade over the course of the rollout. These results corroborate the expected EE and EL task behaviors. Similarly, the case 2 results follow what is expected from the design of the problem. The $j_i^{\mathrm{T}}(k)$ and $j_i^{\mathrm{G}}(k)$ costs are low initially but progressively increase during the rollout. For both cases 1 and 2, the JP task costs increase over the movement due to the choice of goal position. Looking at $j_i^{\mathrm{TP}}(k)$ for each task, it can be observed that the tracking costs tend to dominate due to the initial overshoot in both cases. With each $j_i^{\mathrm{TP}}(k)$ calculated, the total task performance cost, j^{TP} can be evaluated.

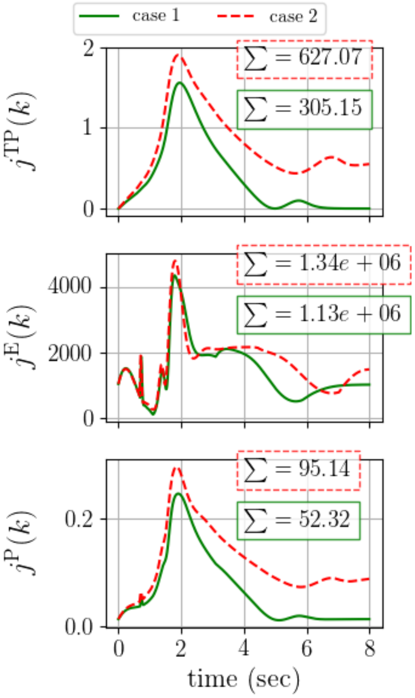


Figure 7.4 – Evolution of the total task performance cost, $j^{\mathrm{TP}}(k)$ (top), energy cost, $j^{\mathrm{E}}(k)$ (middle), and total performance cost, $j^{\mathrm{P}}(k)$ (bottom), for cases 1 and 2.

Task Performance Cost At the top of Fig. 7.4, $j^{\mathrm{TP}}(k)$ for both cases is plotted. Again, the evolution of $j^{\mathrm{TP}}(k)$ is plotted over time with,

$$j^{\mathrm{TP}}(k) = \sum_{i=1}^{n_{\mathrm{task}}} w_i j_i^{\mathrm{TP}}(k), \tag{7.2}$$

where n_{task} is the number of tasks, 3 in this case, and w_i are the task weights. The expected behaviors in task performance can be observed for both cases. Despite the larger performance costs from the JP task, its small weight minimizes its impact on j^{TP}, which is largely dominated by the performance cost of the EE task, $j_{\mathrm{EE}}^{\mathrm{TP}}$.

Energy Cost Next the energy cost at each time step,

$$j^{\mathrm{E}}(k) = \|\boldsymbol{\tau}(k)\|_2^2, \tag{7.3}$$

is determined for both cases. The energy costs are plotted in the middle of Fig. 7.4. Both cases produce roughly identical torque profiles and therefore the energy costs are nearly the same. A peak in $j^{\mathrm{E}}(k)$ at the outset of the movement corresponds to the initial overshoot for both cases. As shown in the figure, $j^{\mathrm{E}}(k)$, is orders of magnitude larger than j^{TP} and this must be taken into consideration for the selection of φ in (5.21). For this example, $\varphi = 1e-4$ is chosen.

Performance Cost Finally, with j^{E} and j^{TP}, the performance cost, j^{P}, is computed using (5.24). The evolution of,

$$j^{\mathrm{P}}(k) = \frac{j^{\mathrm{TP}}(k) + \varphi j^{\mathrm{E}}(k)}{t_{\boldsymbol{\pi}}^{\mathrm{end}}}, \tag{7.4}$$

for cases 1 and 2 is presented in Fig. 7.4. With the given φ, $j^{\mathrm{P}}(k)$ is guided mostly by $j^{\mathrm{TP}}(k)$ with the exception of an initial peak. Nevertheless the overall curves for both cases reflect the expected performance profiles based on the design of the tasks and the trajectories observed. Both cases start with a small $j^{\mathrm{P}}(k)$, and then see a sharp increase due to overshooting the task goal positions. For case 1, $j^{\mathrm{P}}(k)$ returns to a low value, near its starting $j^{\mathrm{P}}(k)$, as the tasks converge successfully to their goal positions. On the other hand, case 2 shows a decreasing $j^{\mathrm{P}}(k)$ as the overshoot is corrected but never gets back to its initial $j^{\mathrm{P}}(k)$ value, as is expected with the incompatible goals.

7.2.2 Comparing to Model-Based Metrics

We now compare the model-free costs with model-based metrics. In Fig. 7.5, two task compatibility metrics, κ^{NR} and κ_A^{NR}, and one task feasibility metric, ϕ^{EE^*}, are plotted for both cases. Looking at the first compatibility metric, κ^{NR} in Fig. 7.5, it can be seen that the tasks produced by the case 1 policy are more compatible over the course of the movement than those of case 2. What is more, the case 1 κ^{NR} values move back to a value near 1.0, indicating a return to compatibility, which coincides with attainment of the goal positions; however, the case 2 values remain low, indicating that the tasks never become compatible, which is a consequence of the chosen policy parameters for case 2. Similarly the κ_A^{NR} metrics in Fig. 7.5, show that the case 1 policy produces tasks which are more compatible with the dynamics than the case 2 policy, albeit the difference is less flagrant than that of the κ^{NR} metric.

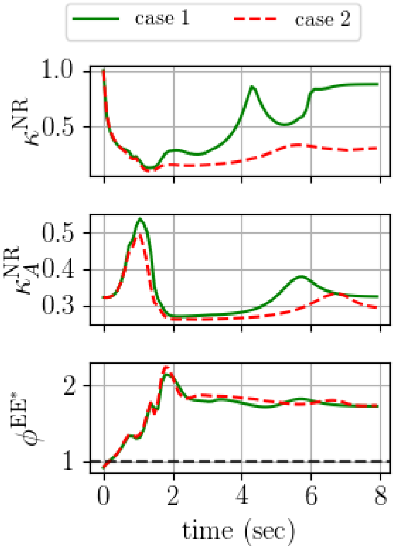


Figure 7.5 – Model-based task compatibility and feasibility metrics from Chapter 3.

The feasibility metric, ϕ^{EE^*} in Fig. 7.5, shows that both policies produce prioritized optima, $\boldsymbol{\chi}^*$, (see Sec. 4.2.2) which are dynamically infeasible according to the feasibility ellipsoid. However, as explained in Sec. 6.6, the feasibility ellipsoid poorly approximates the constraint set near $\boldsymbol{\chi}^*$ in this example because it lies near the vertex of two constraint sets. Knowing that these tasks are therefore feasible, despite the ϕ^{EE^*} measures, one can conclude that the performance cost is high for case 2 because the task objectives are incompatible, which is the design of the scenario.

Combining all of the information gleaned from the metrics, it is fairly clear, without even observing the rollout, that the policy in case 1 produces better task compatibility than that of case 2. This confirms the analyses of the model-free performance cost, which is determined without the complex computations and model understanding needed by the model-based metrics. The model-based metrics do, however, provide more insight into the reason why the performance cost is higher for the case 2 policy.

7.3 Experimental Setup: TCFM Validation

With the model-free cost functions validated, we now apply the TCFM algorithm to case 1. For this experiment, the goal is to use TCFM to modify the EE and elbow trajectories to maximize task compatibility and feasibility as well as attain their original goal positions. For clarity, the task set and constraints for this experiment are shown in Tables 7.5 and 7.6.

Table 7.5 – Task set for the TCFM proof of concept problem. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight** w	**Gain** K_p
Joint Position	All DoF	6	0.0001	10
Cartesian	EE	3	1.0	10
Cartesian	Elbow	3	1.0	10

Table 7.6 – Constraints for the TCFM proof of concept problem.

Constraint Type	DoF/Link	Dimension
Joint Limit	All DoF	6×2
Torque Limit	All DoF	6×2

For case 1, the policy parameterization, $\boldsymbol{\rho}(\boldsymbol{\theta})$, is initially composed of the two minimum jerk trajectory generators and a joint position set point. Since their starting positions are constant, the policy parameter vector consists of the goal positions of the tasks,

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\lambda}_{\text{EE}}^{\text{goal}} \\ \boldsymbol{\lambda}_{\text{EL}}^{\text{goal}} \\ \boldsymbol{\lambda}_{\text{JP}}^{\text{goal}} \end{bmatrix} . \tag{7.5}$$

This gives the policy structure depicted visually in Fig. 7.6.

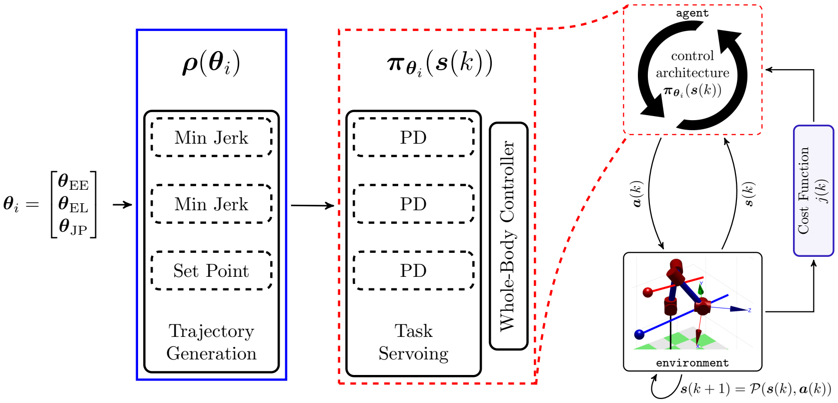

$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{\theta}_{\text{EE}} \\ \boldsymbol{\theta}_{\text{EL}} \\ \boldsymbol{\theta}_{\text{JP}} \end{bmatrix}$
$\boldsymbol{\rho}(\boldsymbol{\theta}_i)$
Min Jerk
Min Jerk
Set Point
Trajectory Generation
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
PD
PD
PD
Task Servoing
Whole-Body Controller
agent
control architecture
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
$\boldsymbol{a}(k)$
$\boldsymbol{s}(k)$
Cost Function
$j(k)$
environment
$\boldsymbol{s}(k+1) = \mathcal{P}(\boldsymbol{s}(k), \boldsymbol{a}(k))$

Figure 7.6 – The model-free policy and its parameterization for the example problem studied here. This figure is adapted from Fig. 5.2

In this experiment, we are interested in maximizing the compatibility and feasibility of these tasks and to do so we focus on the EE and elbow tasks. These principal tasks are highlighted in Table 7.5. Because the objective is for the tasks to attain their goal positions, $\boldsymbol{\theta}$, as defined by (7.5), should not be modified. "Adjustable" parameters must therefore be introduced to these tasks in order to modify them.

7.3.1 Reparameterizing the Tasks

In order to have some free parameters with which to modify the task trajectories, the EE and elbow tasks are reparameterized by sampling the trajectories at their start, middle, and end times. The start and end samples contain the start and goal positions and times, respectively, and the middle sample is the new "adjustable" parameter. From these samples, the tasks are reparameterized by the new waypoints, Λ, and time vectors, $\boldsymbol{t}^{\lambda}$, which consist of,

$$\Lambda = \begin{bmatrix} \boldsymbol{\lambda}_{\text{start}} & \boldsymbol{\lambda}_{\text{middle}} & \boldsymbol{\lambda}_{\text{goal}} \end{bmatrix} \tag{7.6}$$

$$\boldsymbol{t}^{\lambda} = \begin{bmatrix} t_{\text{start}} & t_{\text{middle}} & t_{\text{goal}} \end{bmatrix} . \tag{7.7}$$

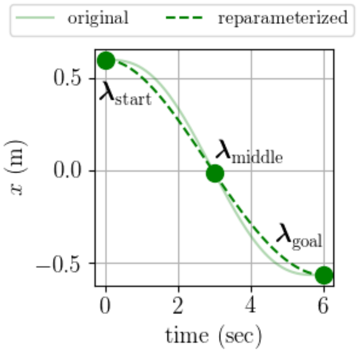


Figure 7.7 – Example of the reparameterized task. This graph shows the x dimension of the EE task trajectory.

From these new waypoints and times, the task trajectories are reconstructed using clamped splines. This process is depicted for the x-axis of the EE task in Fig. 7.7. Thus, the trajectory generators are now clamped splines and the new parameterized policy function, $\boldsymbol{\rho}(\boldsymbol{\theta})$, is depicted in Fig. 7.8. The policy parameter vector is then composed of the EE and elbow middle waypoints,

$$\boldsymbol{\theta}_{i=0} = \begin{bmatrix}\boldsymbol{\theta}_{\text{EE}}\\ \boldsymbol{\theta}_{\text{EL}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{\lambda}^{\text{middle}}_{\text{EE}}\\ \boldsymbol{\lambda}^{\text{middle}}_{\text{EL}}\end{bmatrix}, \tag{7.8}$$

and no longer the joint position task waypoints. For clarity, however, the joint position task parameters, $\boldsymbol{\theta}_{\text{JP}}$, are shown in Fig. 7.8 to indicate that they are still part of the policy but since they remain constant they are no longer part of the policy parameterization, $\boldsymbol{\theta}$. Given the initial policy task trajectories, and following this reparameterization procedure, produces the following parameter vector for the original policy,

$$\boldsymbol{\theta}_{i=0} = \big[\underbrace{-0.013 \quad -0.195 \quad -0.014}_{\boldsymbol{\theta}^{\top}_{\text{EE}}} \quad \underbrace{0.167 \quad -0.131 \quad 0.291}_{\boldsymbol{\theta}^{\top}_{\text{EL}}}\big]^{\top}. \tag{7.9}$$

The TCFM algorithm is then parameterized by $\boldsymbol{\theta}$ from (7.9), and generated by the clamped spline generators.

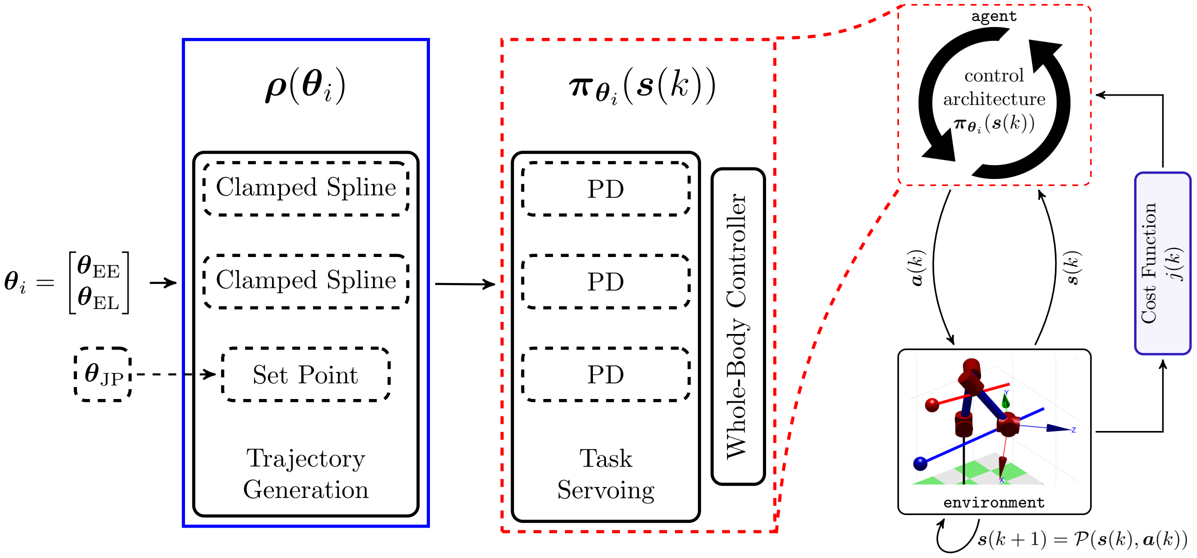

$\boldsymbol{\theta}_i = \begin{bmatrix}\boldsymbol{\theta}_{\text{EE}}\\ \boldsymbol{\theta}_{\text{EL}}\end{bmatrix}$
$\boldsymbol{\theta}_{\text{JP}}$
$\boldsymbol{\rho}(\boldsymbol{\theta}_i)$
Clamped Spline
Clamped Spline
Set Point
Trajectory Generation
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
PD
PD
PD
Task Servoing
Whole-Body Controller
agent
control architecture
$\boldsymbol{\pi}_{\boldsymbol{\theta}_i}(\boldsymbol{s}(k))$
$\boldsymbol{a}(k)$
$\boldsymbol{s}(k)$
Cost Function
$j(k)$
environment
$\boldsymbol{s}(k+1) = \mathcal{P}(\boldsymbol{s}(k), \boldsymbol{a}(k))$

Figure 7.8 – The reparameterized policy function uses clamped splines to generate the trajectory and allows “adjustable” parameters (waypoints) to be added to the EE and elbow tasks. These parameters permit TCFM to optimize the tasks. The joint position task parameters, $\boldsymbol{\theta}_{\text{JP}}$, are shown to indicate that they are remain a part of the policy but since they are constant they are no longer part of the policy parameterization, $\boldsymbol{\theta}$.

7.3.2 TCFM Cost Function and Update Strategy

With these parameters, the performance cost, j^{P}, is used as the TCFM cost function and computed as,

$$j_i = \mathrm{cost}\left(\{\mathcal{S}, \mathcal{A}\}_i\right) = j^{\mathrm{P}} = \frac{j^{\mathrm{TP}} + \varphi j^{\mathrm{E}}}{t^{\mathrm{end}}_{\boldsymbol{\pi}}} = \frac{\left(j^{\mathrm{TP}}_{\mathrm{EE}} + j^{\mathrm{TP}}_{\mathrm{EL}} + j^{\mathrm{TP}}_{\mathrm{JP}}\right) + \varphi j^{\mathrm{E}}}{t^{\mathrm{end}}_{\boldsymbol{\pi}}}, \tag{7.10}$$

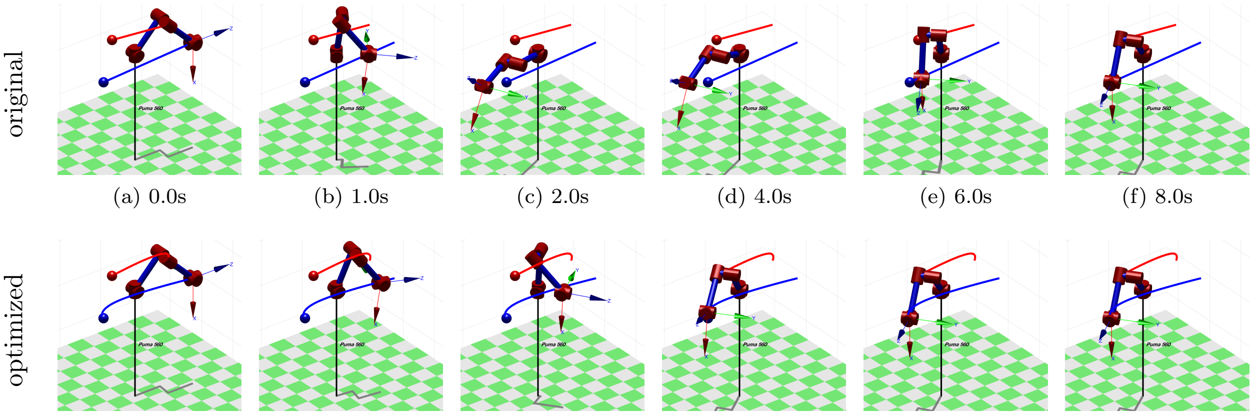
with $\varphi = 1e - 4$ as in the previous experiment, and $t^{\mathrm{end}}_{\boldsymbol{\pi}} = 8.0\mathrm{s}$, the rollout duration. Therefore, all of the tasks are used in the calculation of j_i, even though only the EE and elbow tasks are used in its parameterization. The cost of the original policy, $j_0 = j^{\mathrm{P}}_0$, is used as the scaling factor in (5.25), for the TCFM rollouts.

The update strategy used for the TCFM is CMA-ES. The initial optimization variable variance, which is required by CMA-ES to generate a first estimate of the covariance matrix, is set using,

$$\sigma_{\boldsymbol{\theta}} = 2\mathrm{var}(\boldsymbol{\theta}_{i=0}), \tag{7.11}$$

and the CMA-ES function tolerance for convergence is set arbitrarily to 0.001.

7.4 Results: TCFM Validation



(a) 0.0s (b) 1.0s (c) 2.0s (d) 4.0s (e) 6.0s (f) 8.0s

(g) 0.0s (h) 1.0s (i) 2.0s (j) 4.0s (k) 6.0s (l) 8.0s

Figure 7.9 – Time-lapse of the original and TCFM optimized policies. As can be seen from the images, the optimized policy better tracks its reference trajectories (the blue and red lines) than the original policy and arrives at its goal positions in 6 seconds rather than the original 8 seconds.

Figure 7.9 shows a time-lapse of the original policy, and the optimized policy found used TCFM. The optimized policy parameters are,

$$\boldsymbol{\theta}^{*} = \Big[\underbrace{-0.412 \quad -0.260 \quad 0.134}_{\boldsymbol{\theta}^{\top}_{\mathrm{EE}}} \quad \underbrace{0.175 \quad -0.258 \quad 0.460}_{\boldsymbol{\theta}^{\top}_{\mathrm{EL}}}\Big]^{\top}. \tag{7.12}$$

These parameters are found after 104 rollouts.

The first thing to notice is that the optimized policy never shows any unstable or undesirable behavior like the original policy, which initially overshoots the goal positions. In fact, the optimized policy seems to track the reference task trajectories (indicated by the red and blue lines) well throughout the motion. Unlike the original policy, this smooth and stable tracking allows the robot to reach its goal positions (indicated by the red and blue spheres) in 6 seconds, which corresponds to the expected policy duration. The original policy takes the full rollout duration, 8 seconds, to reach the goal positions.

7.4.1 Trajectory Tracking

To begin the validation of TCFM, the trajectories of the original and optimized policies are examined. The real, denoted “real”, trajectories from the motion execution are the dark lines and their reference, denoted “ref”, trajectories are the lighter lines of the same color and style. The original policy is the same from case 1 in Sec. 7.1. Its task trajectories are indicated by the green lines. The optimized policy task trajectories are indicated by the dashed blue lines.

Looking at Fig. 7.10, it is clear that the optimized trajectories are tracked with higher accuracy than the original trajectories. The exception to this for both the original and optimal trajectories is the y-axis tracking. For this particular movement, it is in the y-axis that the tasks are least compatible because in the middle of the motion the y distance between the reference trajectories for the EE and elbow tasks is smaller than the elbow to EE link length of the robot. Nevertheless, the tracking is much improved with the optimized policy, and this can be confirmed by evaluating the various model-free cost functions.

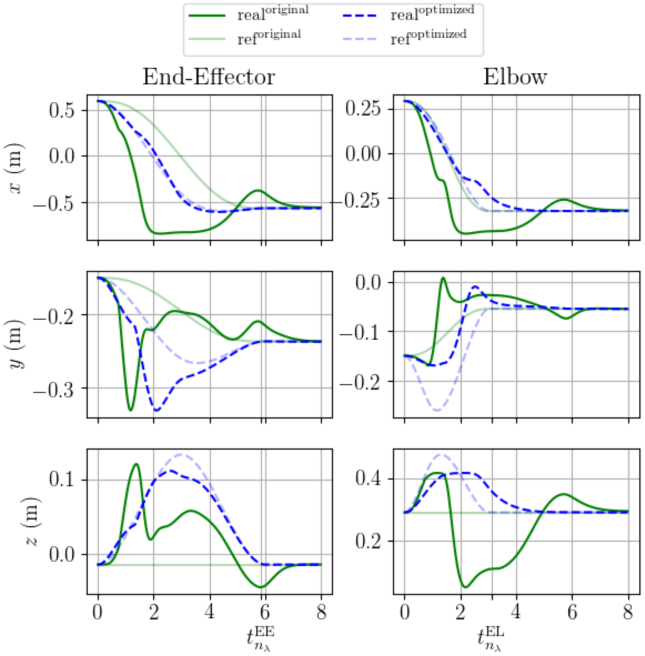


Figure 7.10 – Real and reference values for the EE and elbow task trajectories for the original and optimized policies. The original task trajectories are in green and the optimized task trajectories are in dashed blue. The reference positions are indicated by the lighter lines and the real positions by the darker lines.

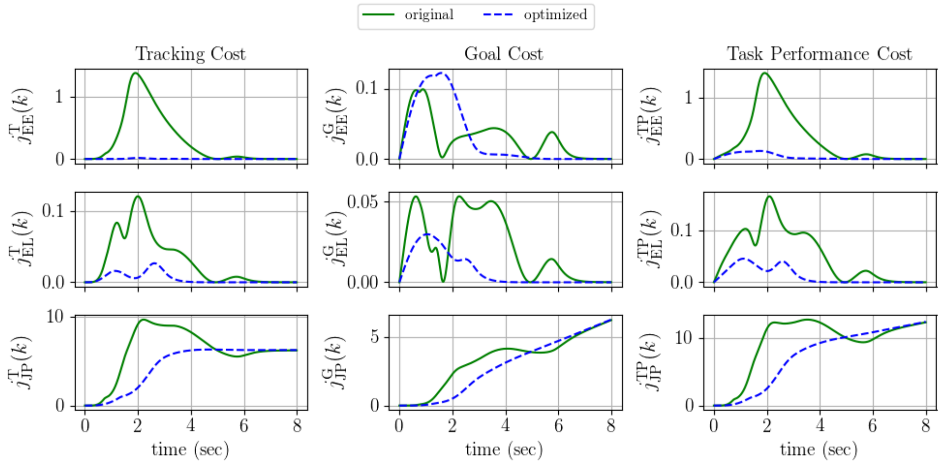


Figure 7.11 – Time evolution of the tracking, $j_i^{\mathrm{TP}}(k)$, goal, $j_i^{\mathrm{TP}}(k)$, and task performance, $j_i^{\mathrm{TP}}(k)$, costs for each task in the example problem from Sec. 6.6 is used. The costs are calculated as the area under the curves, but looking at their evolution allows the task performance to be analyzed.

7.4.2 Cost Analysis

Looking now at the various model-free tracking, goal, and task performance costs, for each of the tasks, it is clear that the tracking is dramatically improved by the optimized policy. In Fig. 7.11, it can be seen that all of the component task performance costs are improved by the optimized policy. With the exception of the joint position task, the tracking costs are essentially null. Likewise, the goal costs are diminished across the board. This produces task performance costs which are significantly smaller than those of the original policy.

In Fig. 7.12, the total task performance, energy, and total performance costs, are plotted for both the original and optimized policies. It is clear from these graphs that the optimized policy improves task performance, reduces energy consumption, and therefore improves the overall motion performance. The result is a set of tasks which are executed as they are planned and a motion which displays behavior which matches what would be expected from the task design.

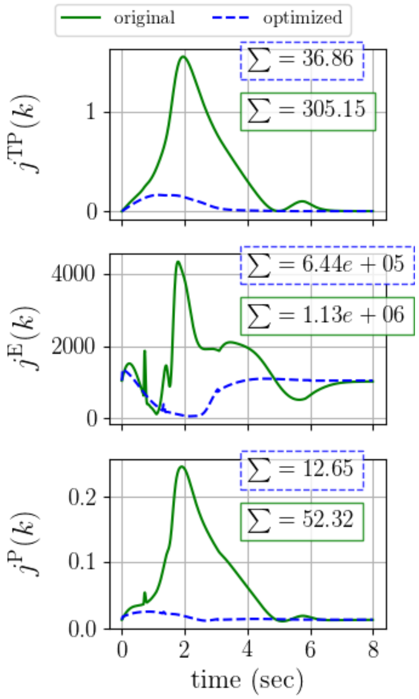


Figure 7.12 – Evolution of the total task performance cost, $j^{\mathrm{TP}}(k)$ (top), energy cost, $j^{\mathrm{E}}(k)$ (middle), and total performance cost, $j^{\mathrm{P}}(k)$ (bottom), for original and optimized policies.

Cost Curves

Looking at the cost curves for the TCFM in Fig. 7.13, shows how the algorithm gradually reduces the performance cost. On the left is the performance cost curve, and on the right are its component costs curves plotted separately. Using the policy parameterization and CMA-ES parameters described here, the TCFM algorithm is not sample efficient at all, but it does minimize the performance cost and produces policies which better match the initial planned intent of the tasks.

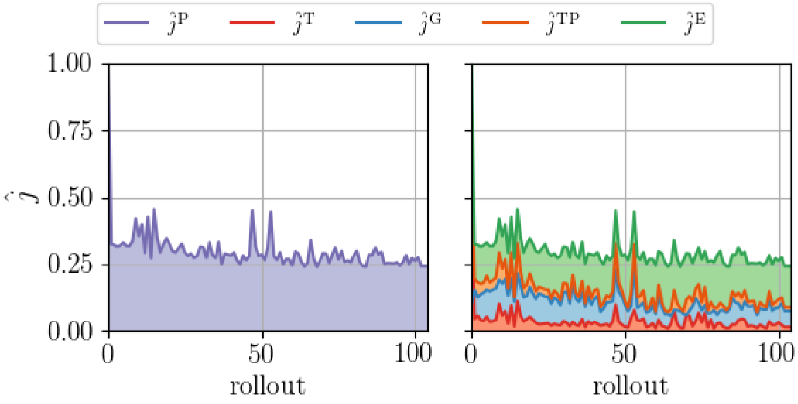


Figure 7.13 – (left) scaled performance cost curve for the TCFM in the proof of concept example. (right) the same curve but plotted by its component scaled costs.

Initially the goal and tracking costs (i.e. the task performance costs) account for the majority of the total performance cost. As the TCFM algorithm updates the policies, the task performance costs are diminished and the energy costs start to account for the majority of the total performance costs. Near the end of the optimization the algorithm is searching for policies which minimize the energy more so than the task performance costs because they account for less than 50% of the total performance cost. Despite its sample inefficiency, the TCFM algorithm effectively minimizes the total performance cost of the policy, but it remains to be seen if this truly has an impact on the task compatibility and feasibility.

7.4.3 Comparaison with Model-Based Metrics

Looking at the model-based task compatibility and feasibility metrics in Fig. 7.14, the standard, κ^{NR} (top), and augmented, κ_A^{NR} (middle), Nuclear norm Ratios are plotted along with the Ellipsoid Evaluation metric, ϕ^{EE^*} (bottom). Again, the closer κ^{NR} is to 1.0, the more compatible the tasks are, and this holds for κ_A^{NR} as well. Here, we can clearly see that the optimized policy increases the κ^{NR} and κ_A^{NR} values throughout the movement. Both the original and optimized policies finish with the same κ^{NR} and κ_A^{NR} values because they finish in the same state, but the optimized policy increases the compatibility throughout the movement.

Looking at feasibility, the ϕ^{EE^*} values tell us how far the prioritized optimum, $\boldsymbol{\chi}^*$, is outside the feasibility ellipsoid. Any value greater than 1.0 indicates that $\boldsymbol{\chi}^*$ is outside of the feasibility ellipsoid. However, for this example we know this to be overly conservative (see Sec. 6.7.2). Nevertheless, it is clear that the optimized policy, although shown as infeasible according to this metric, is more feasible than the original policy throughout the motion. This is further supported by Fig. 7.15, where the distance between the prioritized optimum, and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$, for the original and optimized policies is plotted. This graph shows where the prioritized optimum violates the inequality constraints, and is therefore infeasible. While both policies have brief moments of infeasibility in the beginning of the movement, the optimized policy remains closer to the feasible set and is therefore more feasible in this moment than the original policy when it violates the inequality constraints.

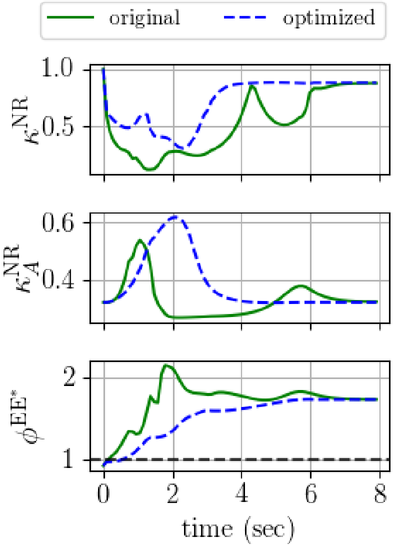


Figure 7.14 – Model-based task compatibility and feasibility metrics from the original and optimized policies.

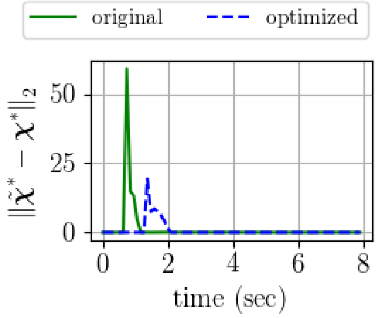


Figure 7.15 – Distance between the prioritized optimum, $\boldsymbol{\chi}^*$, and the controller optimum, $\tilde{\boldsymbol{\chi}}^*$, for the original and optimized policies. Any nonzero distance means the prioritized optimum violates the inequality constraints.

7.5 Conclusions

From these tests, we can confirm that by minimizing the performance cost of a motion the task compatibility and feasibility is maximized. While this implementation is sample inefficient, TCFM is proven to produce compatible and feasible motions according to the model-based metrics designed to measure these specific phenomena. This means that the TCFM algorithm can iteratively improve control policies to achieve desirable overall behaviors which better match the original intent of the planned movement. Now that TCFM has been validated to maximize task compatibility and feasibility, we apply it to a complex humanoid.

Chapter 8

Demonstrating TCFM on Complex Problems

These experiments demonstrate the ability of TCFM to solve complex task incompatibilities and infeasibilities on a humanoid robot through a series of simulated scenarios. Here, Dynamic Movement Primitives (DMPs) are used for policy parameterization and Policy Improvement though Black-Box optimization (PI^{BB}) is used as the TCFM update strategy. Two task trajectories are optimized simultaneously to maximize task compatibility and feasibility. The results show that the TCFM algorithm can successfully maximize task compatibility and feasibility on a humanoid in simulation. However, the chosen parameterization and update strategy require many rollouts to learn an optimal policy.

8.1 Experimental Setup

For each test case studied, a simulation of the iCub humanoid robot is used. In these cases, three principal tasks are specified. The first is a standing task which maintains the CoM at a specific position at all times. This task is designed to keep the iCub from falling over, but since the CoM state is not servoed by a balancing controller (e.g. ZMP), it provides little guarantee that the robot will not lose balance. The other two tasks are reaching tasks associated with reference frames in the palms of the iCub's left and right hands. These are the tasks which are used to generate the different test cases, and consequently whose parameters are optimized in the TCFM. All three tasks have equal priorities, 1.0, in the whole-body controller. Additionally, a low-weight joint position task for all the DoF is used.

The iCub is simulated to be standing on a hard flat surface, and contact constraint sets at the feet are used to model this interaction in the whole-body controller. These constraints are modeled as friction contact points at the four corners of each foot. Each contact introduces 3 equality and 3 inequality constraints (see Sec. 2.3.4), so the dimension for a single contact constraint is 6 and there are 4 contacts per foot. Tables 8.1 and 8.2 list these tasks and constraints as well as their parameters. The hand tasks are initially provided as straight line minimum jerk trajectories from the hands' starting positions to two goal points, indicated by the red and green spheres, for the right and left hands respectively. These spheres can be seen in Fig. 8.1. The joint position task uses the initial posture as its set point.

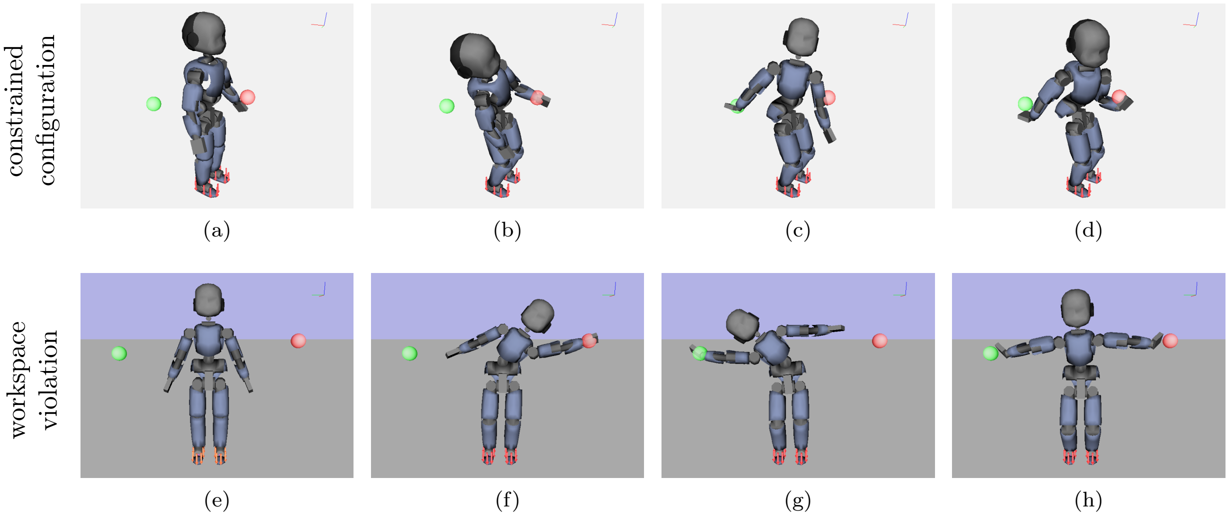
Here, the hand tasks are those of critical importance to the designed whole-body behaviors. All of the

Table 8.1 – Task set for test cases 1 and 2 of TCFM with DMPs and $\mathrm{PI}^{\mathrm{BB}}$. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight** w	**Gain** K_p
Joint Position	All DoF	25	10^{-5}	5
Cartesian	CoM	3	1	1.0
Cartesian	Right Hand	3	1.0	70
Cartesian	Left Hand	3	1.0	70

Table 8.2 – Constraints for test cases 1 and 2 of TCFM with DMPs and $\mathrm{PI}^{\mathrm{BB}}$.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×4
Contact Set	Right Foot	6×4



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 8.1 – (a) - (d) show the constrained configuration scenario and (e) - (h) show the workspace violation scenario. (a) & (e) starting positions. (b) & (f) left hand + standing tasks end configurations. (c) & (g) right hand + standing tasks end configurations. (d) & (h) end configurations of left hand, right hand and standing task combinations where the robot cannot attain both goal positions (spheres) simultaneously.

other tasks exist to make sure these reaches are accomplished. Accordingly, only the hand task parameters are used in the policy parameterization of these experiments, and all of the other task parameters are considered fixed. Given these straight line minimum jerk trajectories, the first step in the TCFM is the parametrization of the original tasks by DMPs. The original tasks considered in this study are all operational-space tasks, but it is possible to parameterize joint-space tasks with DMPs. The forcing terms of each DMP are trained to approximate each task through *Locally Weighted Regression* (LWR) and provide parameterized models of the task DoF. The forcing term slopes and offsets, $\boldsymbol{a}$, $\boldsymbol{b}$, for both the left and right hand DMPs, are used as the policy parameters,

$$\boldsymbol{\theta}_i = \begin{bmatrix} \boldsymbol{\theta}_{\text{left}} \\ \boldsymbol{\theta}_{\text{right}} \end{bmatrix}. \tag{8.1}$$

These parameters, $\boldsymbol{\theta}_i$, are then used to generate the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$ by integrating the DMP trajectory generators in $\boldsymbol{\rho}(\boldsymbol{\theta}_i)$ to get the task reference values. Each DMP forcing term consists of 8 kernels and therefore 8 slopes and offsets thus making $\boldsymbol{\theta}_i \in \mathbb{R}^{32}$. This $\boldsymbol{\pi}_{\boldsymbol{\theta}_i}$ is then rolled out in simulation using the XDE physics simulator and environment described by [Merlhiot et al., 2012; Sovannara, 2013]. With the state and action data, $(\{\mathcal{S}, \mathcal{A}\}_i)$, from the rollout, the cost function is calculated. Here the cost function is only the task performance cost, j^{TP}, for the left and right hand tasks,

$$j_i = \text{cost}\left(\{\mathcal{S}, \mathcal{A}\}_i\right) = \frac{j^{\text{TP}}_{\text{left}} + j^{\text{TP}}_{\text{right}}}{t^{\text{end}}_{\boldsymbol{\pi}}}, \tag{8.2}$$

where $j^{\text{TP}}_{\text{left}}$ and $j^{\text{TP}}_{\text{right}}$ are determined using (5.22) from Sec. 5.6.1. With $j_i = j^{\text{TP}}_i$ calculated for the hand tasks, the PI^{BB} update strategy is used to determine the next $\boldsymbol{\theta}_{i+1}$ to execute. The PI^{BB} update requires a batch of parameter and cost samples to choose the next optimal parameters to test. In these experiments, one update of PI^{BB} corresponds to 15 rollouts. The TCFM is iterated until a convergence criterion is met or a maximum number of 300 rollouts is exceeded.

8.1.1 Test Case 1: Constrained Configuration

The first test case in this experiment is designed such that the iCub is forced into a constrained configuration, as shown in Fig. 8.1d, which prevents either of the hand tasks from being accomplished, i.e. reaching their goal positions. The iCub starts in the posture shown in Fig. 8.1a, and the hand reaching goals are shown by the red and green spheres. Each of these goals is individually attainable, as shown in Fig. 8.1b and Fig. 8.1c. However, when both tasks are executed simultaneously, the trajectories to their goals force the robot into the constrained posture shown by Fig. 8.1d. The durations of the hand task trajectories are both equal to 4.0 seconds. The objective of this test case is to see if TCFM can find task parameters, which allow the robot to attain its hand goal positions by improving the task compatibility and feasibility, through modification of the left and right hand trajectories.

8.1.2 Test Case 2: Workspace Violation

In this second test case, the hand goal positions are outside of the iCub's workspace and therefore the robot can never maintain both goals simultaneously. If both tasks end at the same time, then modification of the DMP trajectories will not render the combination fully compatible because their attractor points are incompatible. Here, one must consider the time component of the reaching problem to find an appropriate solution, i.e. both tasks must be modified and done in some sequence, in order for the robot to meet its objectives. Therefore, the two hand trajectories have different durations, with the right hand trajectory being shorter than the left hand trajectory, 3.0 and 4.0 seconds respectively. To examine the effectiveness of the TCFM in cases where task sequencing is a factor, two examples are studied:

1. **Without Task Deactivation:** The task goal positions are maintained as attractor points even after the end of the duration of the task, therefore the task remains activated.
2. **With Task Deactivation:** Once a hand task has reached the end of its duration and its goal position, within some small margin of error, the task is deactivated (i.e. its QP weight set to zero) and no longer influences the resolution of the whole-body control problem.

The objective of this test case is to demonstrate how time can be exploited within TCFM to find solutions to task incompatibilities, as well as how higher-level concepts like task deactivation can affect the policy rollouts.

8.2 Results

In this section the results for the experimental procedure developed in Sec. 8.1 are presented. The results for the first test case, which explores incompatibilities which lead to a constrained configuration, are presented initially, followed by the results of the second case which looks at a workspace violation scenario.

8.2.1 Test Case 1: Constrained Configuration

When either the right or left hand task is combined with the standing task, the combination is compatible and the reach can be completed; see 8.1b and 8.1c. However, the combination of both left and right hand reaches, along with the standing task, generates an incompatible set of task objectives, 8.1d. This incompatibility is due to the fact that each hand task generates opposing commands for the torso roll and yaw as seen in Fig. 8.2. The result of these conflicts, shown in Fig. 8.1d, is that the hand tasks never achieve their final goal positions and poorly track their reference trajectories. Applying the TCFM to the constrained configuration case results in optimized tasks which reduce the task performance cost from (8.2), and attain the final goal positions for the hand tasks. The evolution of the cost curve for the test cases is presented in Fig. 8.3 and a time-lapse of the movement generated by the optimized tasks is shown in Fig. 8.4. Because the goal positions for the hands are kinematically compatible, i.e. the robot can attain this configuration, the problem is in the way the robot gets to these positions. From the time-lapse shown in Fig. 8.4, the optimized DMPs cause the robot to not follow straight paths to the goal points, but rather to rock the robot torso into a posture which allows the goal positions to be reached.

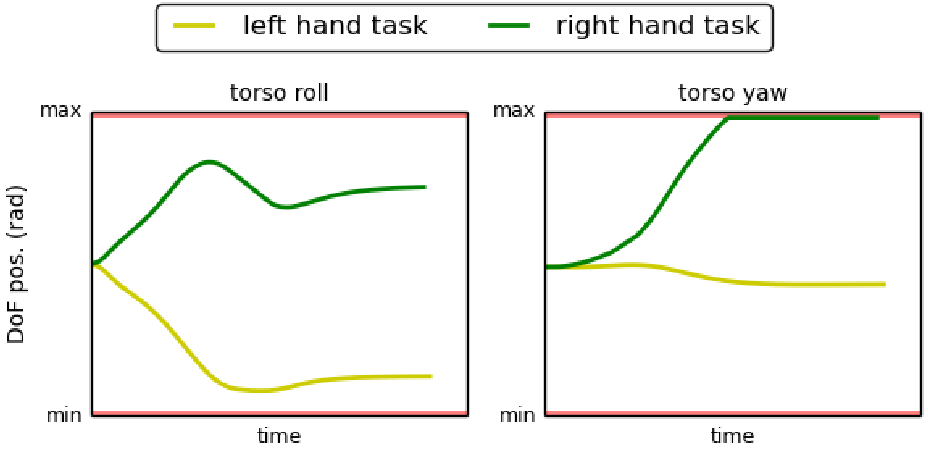


Figure 8.2 – Evolution of torso DoFs during the execution of the left hand + standing and right hand + standing combinations.

8.2.2 Test Case 2: Workspace Violation

In Fig. 8.1f and Fig. 8.1g it can be seen that, when either the left or the right hand tasks are executed with the standing task, the combinations are feasible. However, in Fig. 8.1h the combination of all three tasks generates workspace incompatibilities caused by the fact that the robot's workspace is not large enough to accommodate both trajectories. This is illustrated in Fig. 8.5, where the distance between the tasks at each instant in time while they are active is plotted for the original and optimized sets of tasks. At approximately 2.1 seconds, the distance between the original tasks begins to exceed the maximum workspace of the iCub, and generates task incompatibilities. Since neither of the tasks reaches its goal,

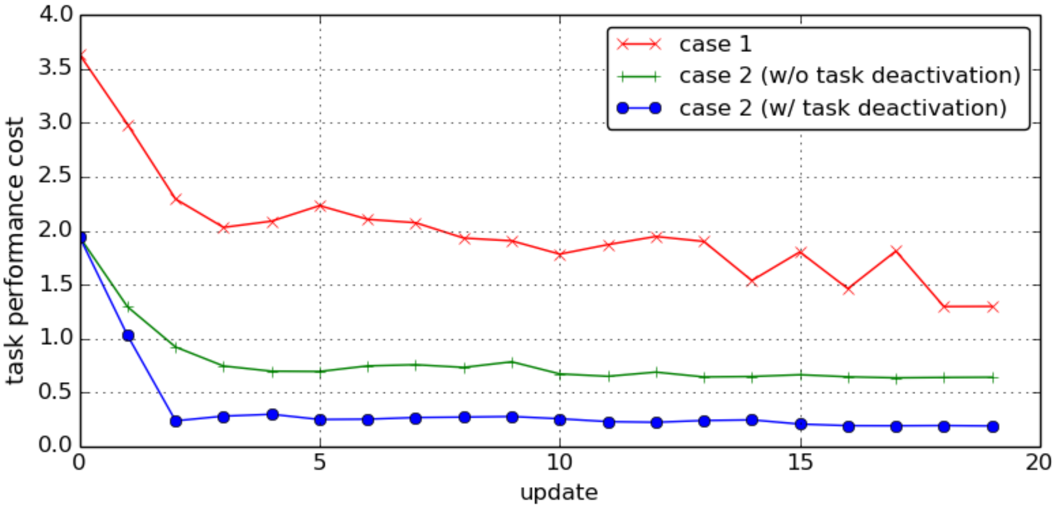
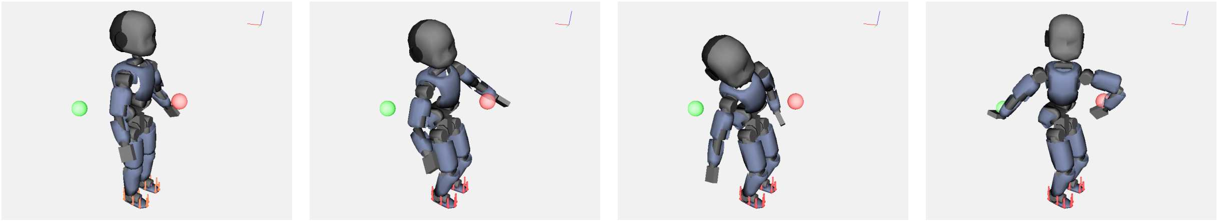


Figure 8.3 – Cost curves for the TCFM optimizations for both test cases. Note that each update corresponds to 15 policy rollouts. For all cases, the TCFM, using DMPs and PI^{BB}, is stopped after 300 rollouts, because the update policy convergence criterion is never attained.



(a) 0.0s (b) 1.5s (c) 2.3s (d) 4.0s

Figure 8.4 – Optimized task combinations found with TCFM. (a) - (d) show a time-lapse of the optimized tasks for test case 1 from Sec.8.1.1.

they cannot be deactivated and whole-body control alone cannot accomplish this task combination. In the following sections, the results of TCFM both without and with task deactivation are presented. Time-lapses of the optimized motions both with and without task deactivation are shown in Fig. 8.6.

Without Task Deactivation

The set of tasks optimized by the TCFM without task deactivation modifies the hand trajectories to respect the workspace limit for as long as possible, and manages to attain the right hand task goal position. Unfortunately, because the right hand task is not deactivated, its goal position remains as a system attractor, and the robot is forced to violate the workspace limit. The overall result of this solution is a set of tasks which reduces the task performance cost (see Fig. 8.3), but cannot attain all of the desired goal positions.

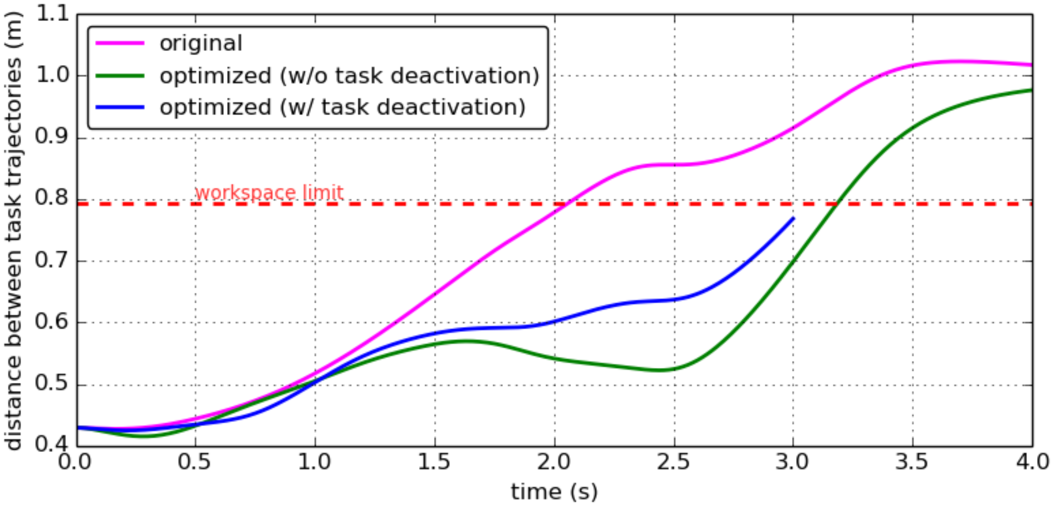
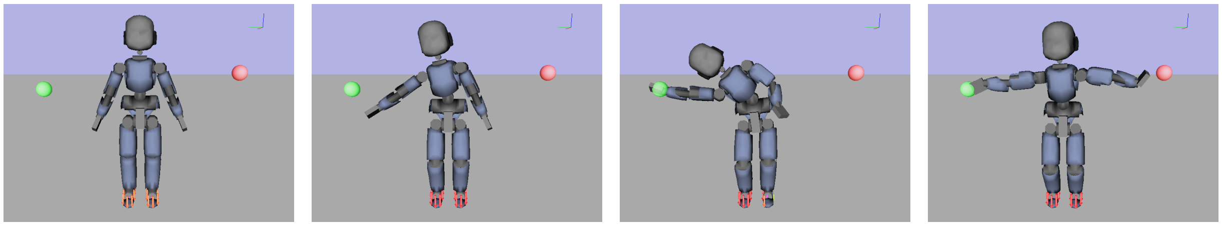
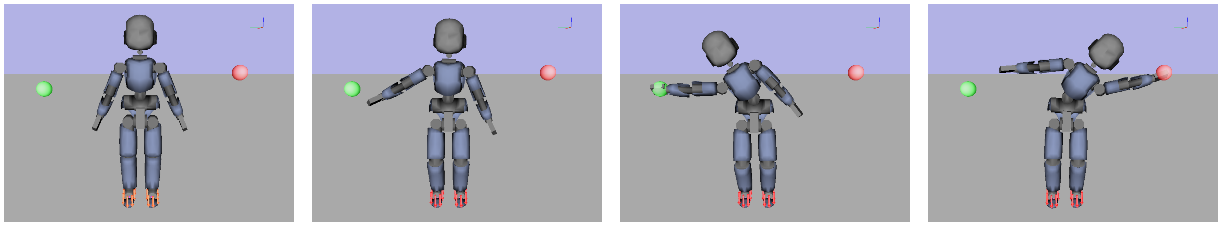


Figure 8.5 – A plot of the distances between the hand task trajectories for each instant in time when they are simultaneously active. The optimized DMPs without task deactivation delay this incompatibility, and the optimized DMPs with task deactivation remove it.



(a) 0.0s (b) 1.7s (c) 2.8s (d) 4.0s



(e) 0.0s (f) 1.7s (g) 2.8s (h) 4.0s

Figure 8.6 – Optimized task combinations found with TCFM. (a) - (h) show two time-lapses of the the optimized tasks for test case 2 described in Sec. 8.1.2. (a) - (d) show the solution found without task deactivation and (e) - (h) show the solution found with task deactivation.

With Task Deactivation

When the tasks are optimized in the TCFM with task deactivation, the overall task performance cost is reduced as shown in Fig. 8.3, and the hand tasks each attain their goal positions as shown in Figs. 8.6g and 8.6h. This is possible because once the right hand goal position is attained, that task is deactivated and no longer contributes to the task performance cost of the execution. This is reflected in Fig. 8.5, where the distance between task curve terminates at 3.0 seconds, when the right hand task is completed.

8.3 Conclusions

These experiments demonstrate that TCFM can effectively optimize incompatible and infeasible tasks on a complex platform, i.e. the iCub humanoid robot. Its major drawback however, is sample inefficiency. Each update of $\mathrm{PI}^{\mathrm{BB}}$ (in this implementation) requires 15 policy rollouts, and convergence based on minimum solution differences is not achieved after 300 rollouts for all cases.

This is the result of two main causes. Firstly, the policy parameterization, $\boldsymbol{\theta} \in \mathbb{R}^{32}$. This is a large space which needs lots of data to explore. Secondly, as a consequence of the point attractor nature of DMPs, the parameters near the end of the movement must be much larger in magnitude to compensate for the attenuating effects of the goal attractor term. These terms are designed to ensure convergence but they create a parameter vector with wildly different scales of values as for the kernels closest to the beginning of the movement to those near the end. This results in a parameter space with cost gradients which are highly biased in the directions of the end kernel parameters, but these parameters have the least effect on the overall movement. The end result is a very inefficient learning process.

Finally, although it cannot be observed in the time-lapses, the resulting behaviors after TCFM seem to spend a lot of time making motions which do not help in the overall whole-body performance. The video shows this in more detail. This is due to the fact that only the task performance cost was used. No terms penalize energy expenditure. So, despite having shown the ability to correct task incompatibilities and infeasibilities, TCFM needs to be refined in the following ways:

1. A more efficient policy parameterization which only uses pertinent task parameters for the TCFM.
2. A more efficient update strategy to make better updates with less data, i.e. fewer rollouts.
3. Integration of an energy cost term to guide the search towards efficient behaviors.

Chapter 9

Improving Sample Efficiency

In the experiments performed in Chapter 8, it is observed that TCFM has the potential to resolve task incompatibilities and infeasibilities on humanoid robots. Unfortunately, the chosen Dynamic Movement Primitive (DMP) parameterization in association with the Policy Improvement though Black-Box optimization (PI^{BB}) update strategy requires a large number of rollouts to achieve optimal task trajectories. This is unacceptable for real robots. To resolve this problem, Gaussian Process Trajectories (GPTs) are explored as a more compact and expressive policy parameterization, and Bayesian Optimization (BO) as a more sample efficient TCFM update strategy. This chapter is divided into two parts. In the first, we look at the effects of using GPTs to generate the task trajectories in a purely reactive setting. The objective is to understand if stochastic policies provide any benefit in terms of task compatibility and feasibility improvement by simply allowing the controller more flexibility in its prioritization scheme. These effects are studied on the same examples presented in Chapter 8, and a new example which brings balancing into play. Having understood the effects of the new policy parameterization on the overall execution, it is then used within the TCFM algorithm, with BO as the update strategy. In this second part, external perturbations and model perturbations are used to show how a variety of complex factors can degrade the overall behavior of the robot. By optimizing the performance cost, the robot can correct for these factors. These perturbations are created to simulate incompatibilities and infeasibilities in a visual and demonstrative way. Through two scenarios, it is shown that TCFM can efficiently maximize task compatibility and feasibility. The effect of deterministic and stochastic policies on TCFM is explored and robustness issues are brought to light.

9.1 Experimental Setup: GPTs as a Policy Parameterization

In this experiment, three simulated scenarios are presented to highlight some common issues encountered when combining multiple incompatible tasks. The first two are similar to those explored in Chapter 8, and the third looks at equilibrium related incompatibilities and infeasibilities.

The iCub humanoid robot is once again simulated in the XDE physics environment [Merlhiot et al., 2012; Sovannara, 2013]. Task completion is characterized as the proximity of the hand task frames to their respective goal locations within a margin of 3.0 cm. More precise margins can be applied without loss of generality. The policies are rolled out using both deterministic, $\boldsymbol{a}(k) = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{s}(k))$ (see Sec. 5.2.3), and stochastic policies, $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{a}(k)|\boldsymbol{s}(k))$ (see Sec. 5.3.2). To simplify the discussion, deterministic policies are referred to as static, and stochastic policies are referred to as variable. This is because both policies use the same weighted priority QP controller described in Chapter 2, and the deterministic policies do not change the controller weights over the course of the movement, while the stochastic policies do vary the weights as described in Sec. 5.3.2.

9.1.1 Test Case 1: Constrained Configuration

In this first test case, three principle tasks are combined which lead the to the robot being forced into a constrained configuration. Such configurations commonly occur on highly redundant systems when multiple tasks require the same DoF. In humanoids, this often occurs due to solicitation of the torso DoF. The first standing task maintains the robot's Center of Mass (CoM) at a constant position with a static weight of 1.0. The joint position, torso orientation, and head orientation tasks are used to keep the robot in an upright posture and avoid unactuated articulations. The two variable weight tasks are associated with the left and right hands, specifically the center of the base of the palms, thus it is these tasks which have GPT. These tasks are defined by trajectories passing through waypoints at the beginning, middle and end of the movements. The task objectives are for them to attain the final waypoints, or goal positions, while passing through the other waypoints. The left and right hand tasks last 6.4s and 6.1s respectively, and are executed with $\beta = 1.0$ (see (5.15)). Tables 9.1 and 9.2 list these tasks and constraints as well as their parameters.

Table 9.1 – Task set for test cases 1 of the GPTs as a policy parameterization experiment. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight** w	**Gain** K_p
Joint Position	All DoF	25	10^{-5}	9
Cartesian	CoM	3	1	36
Orientation	Torso	2	0.001	16
Orientation	Head	2	0.1	16
Cartesian	Right Hand	3	1.0	40
Cartesian	Left Hand	3	1.0	40

Table 9.2 – Constraints for test cases 1 of the GPTs as a policy parameterization experiment

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×4
Contact Set	Right Foot	6×4

9.1.2 Test Case 2: Workspace Violation

The second test case combines the same tasks and constraints described in the first test case in Sec. 9.2.1. However, this time the hand trajectory goal positions are further apart than the maximum workspace of the robot (in this standing configuration). This scenario is designed to represent a typical workspace conflict during picking procedures. The trajectories pass through waypoints at the beginning, middle and end of the movements. The left and right hand tasks trajectory durations are 6.3s and 6.2s respectively, and are executed with $\beta = 1.0$. When one of the hands attains its goal position, that is, within 3.0cm of the final waypoint, that task is deactivated (i.e. the object has been picked). Task deactivation means that it no longer contributes to the control solution, or equivalently, that its weight is set to 0.0.

9.1.3 Test Case 3: Balance Perturbation

In this third test case, the objective is to create a task which is dynamically incompatible with bipedal equilibrium. Equilibrium is managed by trying to keep the CoM position over the center of the Polygon of Support (PoS), which is defined by the convex hull of foot contact sets. The right hand follows a sweeping trajectory from the hand's starting waypoint to its end waypoint - no intermediary waypoints are considered. The right hand task is managed by a GPT. Tables 9.3 and 9.4 list these tasks and constraints as well as their parameters. The right hand task trajectory duration is 8.6s and is executed with $\beta = 10.0$.

Table 9.3 – Task set for test cases 3 of the GPTs as a policy parameterization experiment. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Joint Position	All DoF	25	10^{-5}	9
Cartesian	CoM	3	1	36
Cartesian	CoM	3	1	50
Orientation	Torso	2	0.001	16
Orientation	Head	2	0.1	16
Cartesian	Right Hand	3	1.0	40

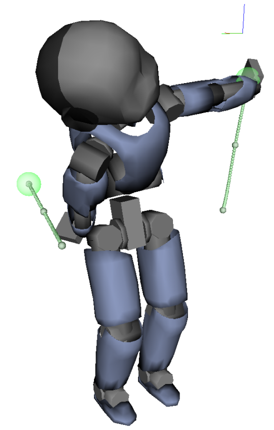
Table 9.4 – Constraints for test cases 3 of the GPTs as a policy parameterization experiment

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×4
Contact Set	Right Foot	6×4

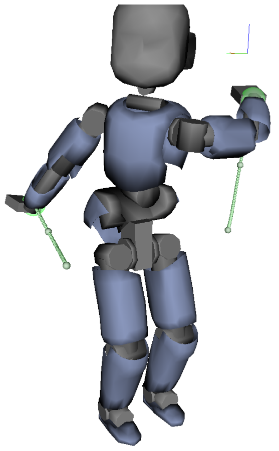
9.2 Results: GPTs as a Policy Parameterization

In this section we provide the results of the scenario simulations described in Sec. 9.1.

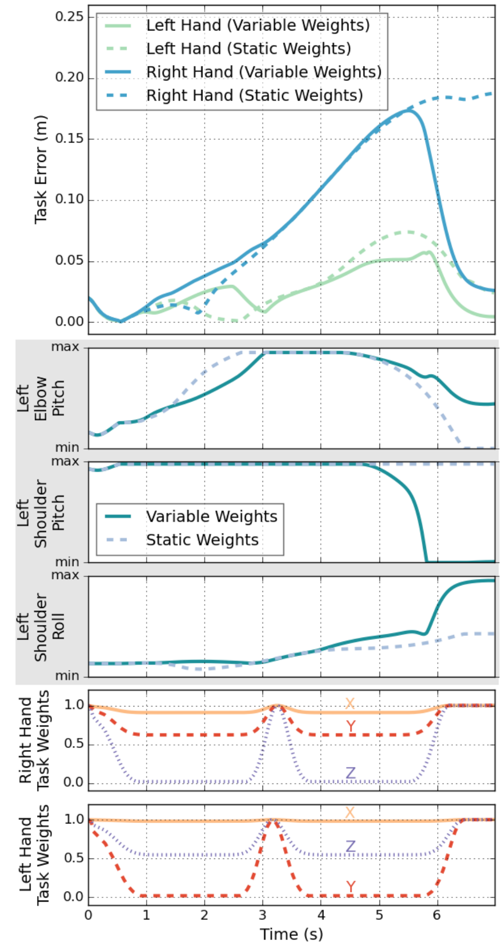
9.2.1 Test Case 1: Constrained Configuration



(a) static



(b) variable



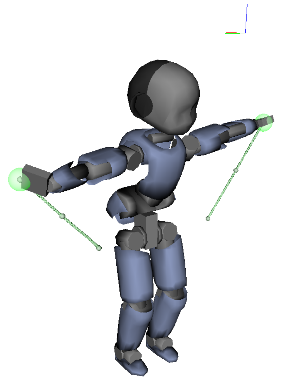
(c)

Figure 9.1 – The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters.

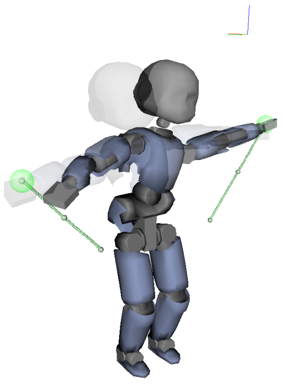
When the two hand tasks are combined with static weights, we can see in Fig. 9.1a that the left hand task achieves its goal location, contrary to the right hand task. This occurs because individually, the hand tasks require the torso to rotate left and right; therefore, when they are combined this DoF is constrained between the two. The arm DoF attempt to compensate for this reduction in redundancy by moving to their limits, and forcing the robot into a constrained configuration. This is shown in the left arm DoF plots in Fig. 9.1c. Consequently, the right hand task is no longer feasible and incurs high task errors at both the middle and goal waypoints due to its combination with the left hand task. This can be observed in the task error plot of Fig. 9.1c. The waypoints along the trajectory are indicated by the peaks in the task weight curves in Fig. 9.1c.

In Fig. 9.1b, the robot has successfully accomplished its tasks through the use of variable weights. By looking at the left arm DoF plots in Fig. 9.1c, we can see that the right hand task weight increases approximately 0.25s prior to the left hand weight, forcing the controller to prioritize its execution and causing the left arm elbow pitch, shoulder pitch and shoulder roll to deviate. These deviations pull the left arm DoF away from their limit values, freeing these articulations for the left hand movement when its weight increases.

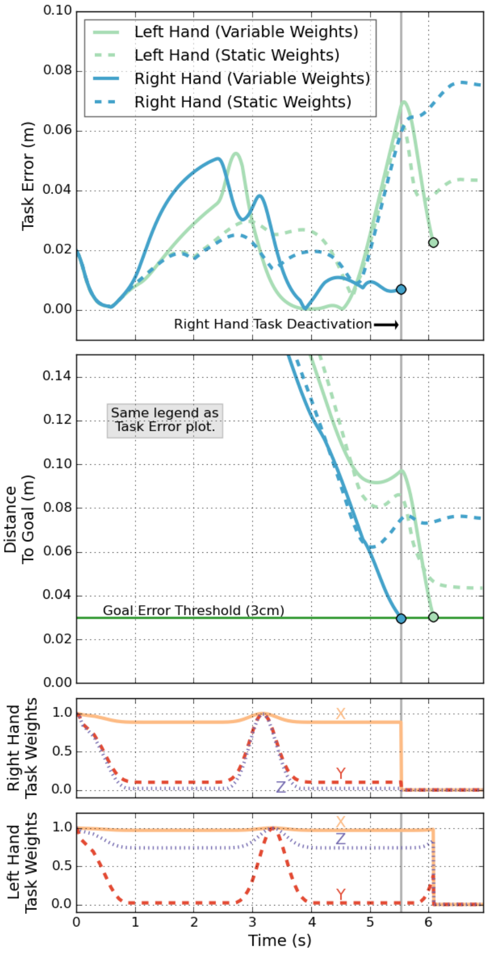
9.2.2 Test Case 2: Workspace Violation



(a) static



(b) variable



(c)

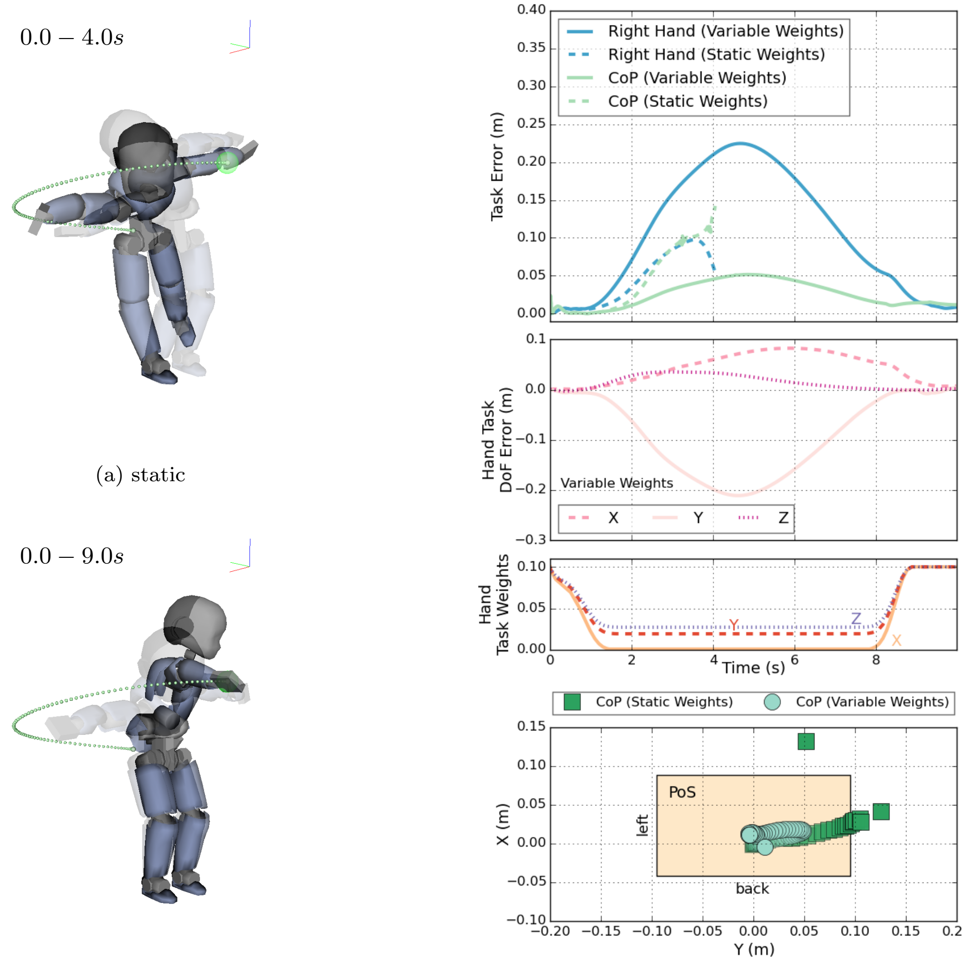
Figure 9.2 – The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters.

Figure 9.2a shows the static execution of the two hand tasks and although the hands seem to reach their goal positions, close inspection of the distance to goal plot in 9.2c shows that they never attain the 3.0cm error threshold limit. As a result, they rest in a local minimum between their two objectives.

When variable weights are applied to the simultaneous execution of the two hand tasks, the robot achieves its right hand goal first, thereby deactivating the right hand task, and then proceeds to finish the left hand task; this is shown in Fig. 9.2b. The instants that the hand tasks are deactivated can be seen in the task error and distance to goal plots, and are indicated by circular markers.

In both the right and left hand movements, the y-axis component develops large errors near the goal locations. The errors are roughly equivalent (see Fig. 9.2c static task error plot) and therefore whichever task has the largest weight (in the y-direction) dominates in the whole-body controller output — the right hand task in this case (see Fig. 9.2c hand task weight plots). Once the right hand task is deactivated, all incompatibilities are removed and the left hand task is able to recuperate its accumulated error and be deactivated as well.

9.2.3 Test Case 3: Balance Perturbation



(a) static

(b) variable

(c)

Figure 9.3 – The constrained configuration scenario. (a) and (b) show the task combination results using static and variable weights respectively. (c) plots provide the simultaneous evolution of various task parameters.

Figures 9.3a and 9.3b show the balance perturbation results. Using static weights for the right hand task results in a loss of balance and ultimately a failure for both tasks; this can be seen in Fig. 9.3a. We can confirm this loss of balance by observing that the *Center of Pressure* (CoP) moves outside of the PoS in Fig. 9.3c. Despite the CoM task being ten times more important than the right hand task, the motion fails because the accumulated error at the apex of the sweeping movement generates large enough accelerations in the y-direction to perturb the balance.

In the variable weight case, we can see in Fig. 9.3b that the robot successfully attains the goal position of the hand task while remaining balanced. The task error plot shows that the right hand task incurs a large amount of error as in the static case, but because this occurs during a period of high variance, this error only partially perturbs the CoM task. The CoM task is deviated somewhat from its goal location in order to compensate for some of the right hand error, but the CoP remains safely within the PoS as shown in Fig. 9.3c.

9.2.4 Conclusions: Effect of GPTs as a Policy Parameterization

From the experiments presented here it is clear that stochastic policies have the potential to improve the robustness of tasks to incompatibilities and infeasibilities, by allowing the controller to allocate resources as necessary depending on the variance of the task trajectories. The results here show that the stochastic policies can reactively get out of constrained configurations, solve workspace conflicts, and avoid balancing incompatibilities. Nevertheless, all of this success hinges on the synchronization of the points of high trajectory variance and high incompatibility and infeasibility. Taking the balancing perturbation scenario for instance, if a waypoint had been placed near the apex of the reaching trajectory then the variance of the right hand task would have gone to near zero, and its weight would have increased to its maximum value. This would have caused the robot to lose balance. Thus, despite their simplicity and interesting properties, stochastic policies are not a panacea for task compatibility and feasibility improvement, because it is clear where they will not work. Therefore, there remains a need for TCFM. On this note, the next section explores how GPTs can be integrated into the TCFM algorithm and used as a compact policy parameterization, which allows the use of a BO update strategy.

9.3 Experimental Setup: TCFM using GPTs and BO

In this experiment, the GPT policy parameterization is used within the TCFM algorithm in concordance with BO as a update strategy to see if sample efficient TCFM is achievable and therefore feasible for real robots. Two test scenarios are explored to study the efficacy of these changes to the TCFM algorithm. For each scenario the TCFM algorithm is run with and without use of the task variance, i.e. stochastic policies.

Here, all simulations are executed in the Gazebo [Koenig and Howard, 2004] environment using the ODE physics engine and a real-time clock. The first test case provides a demonstration of how to use GPTs to parameterize tasks and add "adjustable" parameters (waypoints in this case) which can be used by TCFM to minimize the cost function. In doing so, we observe that a repeated external perturbation, which is used as facsimile for a time-varying Cartesian infeasibility, can be accounted and compensated for. In the second, more complex, test case, a set of dynamically incompatible tasks, actuator limits, and environmental interactions generate a complicated set of task incompatibilities and infeasibilities, which degrade the robot's ability to perform its desired tasks. In both tests, the use of environmental information is explicitly prevented, and only the right hand task is used in the policy parameterization. Here, the total performance cost, j^{P}, is used as the cost function,

$$j_i = \operatorname{cost}\left(\{\mathcal{S}, \mathcal{A}\}_i\right) = j^{\mathrm{P}}. \tag{9.1}$$

Since only the right hand task is used in the policy parameterization, the performance cost consists of,

$$j^{\mathrm{P}} = \frac{j^{\mathrm{TP}}_{\mathrm{right}} + \varphi j^{\mathrm{E}}}{t^{\mathrm{end}}_{\boldsymbol{\pi}}}, \tag{9.2}$$

with $\varphi = \max\left(\|\boldsymbol{\tau}(k)\|_2\right)^{-1}$, the maximum torque norm from the original policy, as the energy scaling factor.

9.3.1 Test Case 1: Operational-Space Infeasibilities

In this toy example, the goal is to illustrate the key components of the task parameterization with GPTs and the usage of BO as a TCFM update strategy. The basic idea of the scenario is to simulate a complicated time-varying infeasibility in the operational-space and use this to provoke infeasibilities in the prescribed tasks. To accomplish this, a physical obstacle is inserted into the path of a right hand reaching task at a specific time. Through the use of an obstacle to introduce task infeasibilities, the TCFM scenario can be closely controlled and studied. Real time-varying constraints for humanoid robots, such as joint position constraints, are difficult to clearly analyze and control. Using an obstacle on the other hand, allows us to isolate the task infeasibility to a known and constant source. To further control the situation, the root link of the robot is fixed to isolate task disturbances to the presence of the obstacle and remove any possible issues due to maintaining balance. The only tasks active in the test scenario are full and partial posture (joint position) tasks, and a right hand position task. The full-body and torso posture tasks are used to keep the robot in an upright posture and avoid unactuated limbs. Only the right hand reaching task is of interest in this study and is therefore the focus of test case 1. These tasks and their parameters are presented in Table 9.5.

Table 9.5 – Task set for test case 1 of TCFM with GPTs and BO. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Joint Position	All DoF	25	10^{-5}	5
Joint Position	Torso DoF	3	10^{-3}	30
Cartesian	Right Hand	3	1.0	20

The robot is given a trajectory for its right hand Cartesian task from the hand's starting position to a point 25cm above it with the following waypoints,

$$\Lambda = \begin{bmatrix} z_{start} & z_{goal} \end{bmatrix} \tag{9.3}$$

$$\boldsymbol{t}^{\lambda} = \begin{bmatrix} t_{start} & t_{goal} \end{bmatrix} \tag{9.4}$$

where $z_{start} = 0.0$m, $z_{goal} = 0.25$m, $t_{start} = 0.0$s, and $t_{goal} = 2.5$s. The x and y axes of the task frame are constrained to force the robot to move in a straight line along the z-axis preventing the robot from moving around the obstacle. The robot must optimize the right hand Cartesian task to avoid an obstacle (compensate for a task incompatibility), of which it has no knowledge or perception. Because the right hand task waypoints cannot be modified, an additional "adjustable" waypoint is inserted into (9.3),

$$\Lambda = \begin{bmatrix} 0.0 & z' & 0.25 \end{bmatrix} \tag{9.5}$$

$$\boldsymbol{t}^{\lambda} = \begin{bmatrix} 0.0 & t' & 2.5 \end{bmatrix}. \tag{9.6}$$

Here, t' is taken as the median time of the task trajectory, 1.25s, and $z' = \xi(t') = 0.12$m. This waypoint is then used as the initial policy parameters,

$$\boldsymbol{\theta}_{i=0} = \begin{bmatrix} z' \\ t' \end{bmatrix} = \begin{bmatrix} 0.12 \\ 1.25 \end{bmatrix}. \tag{9.7}$$

The policy parameters, $\boldsymbol{\theta} \in \mathbb{R}^2$, so the surrogate function mean within the BO may be visualized. The rollout is terminated if the hand has reached its goal location to within 3.0cm or the policy rollout has exceeded 10.0s in total duration. The selected BO bounds are the minimum and maximum waypoint, Λ, and waypoint times, $\boldsymbol{t}^{\lambda}$,

$$\begin{bmatrix} 0.0 \\ 0.1 \end{bmatrix} \leq \boldsymbol{\theta} \leq \begin{bmatrix} 0.25 \\ 2.4 \end{bmatrix}, \tag{9.8}$$

with a small margin of difference in the t' bounds to avoid having two simultaneous waypoints. At exactly 1.0s, the obstacle is inserted into the path of the robot's hand trajectory at $z = 0.12$m; however, this information is not used in any way during the task optimization. This obstacle is a 1.0cm thick flat plate (see Fig. 9.4).

9.3.2 Test Case 2: Moving a Heavy Weight

The aim of this simulation is to show how TCFM can compensate for a complex combination of task incompatibilities and infeasibilities, which impair the overall behavior of the robot. In this example, the robot is standing on a flat surface and maintaining balance, while trying to move a 1.0kg cube with its right arm from its starting point to a target location on a flat table. The feet are not fixed to the ground. The cube is affixed to the right hand and its mass is greater than the maximum effective payload of the iCub's right arm, which is approximately 0.5kg when fully extended. An orientation task is used to maintain the cube's bottom surface parallel to the table top. Balance is achieved through a CoM position task, the objective of which is to maintain the CoM's ground projection in the center of the PoS. A torso Cartesian task along with a full-body posture task keep the robot upright. These tasks and constraints along with their parameters are presented in Tables 9.6 and 9.7.

Table 9.6 – Task set for test case 2 of TCFM with GPTs and BO. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Joint Position	All DoF	25	10^{-5}	5
Cartesian	CoM	3	1.0	50.0
Cartesian	Torso	3	10^{-4}	1.0
Orientation	Right Hand	3	1.0	80
Cartesian	Right Hand	3	0.05	70

Table 9.7 – Constraints for test case 2 of TCFM with GPTs and BO.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25 × 2
Torque Limit	All DoF	25 × 2
Contact Set	Left Foot	6 × 4
Contact Set	Right Foot	6 × 4

The right hand trajectory consists of three waypoints, at the start, middle, and goal of the movement,

$$\Lambda = \begin{bmatrix}\boldsymbol{\lambda}_{start} & \boldsymbol{\lambda}_{mid} & \boldsymbol{\lambda}_{goal}\end{bmatrix} \tag{9.9}$$

$$\boldsymbol{t}^{\lambda} = \begin{bmatrix}t_{start} & t_{mid} & t_{goal}\end{bmatrix}. \tag{9.10}$$

The middle waypoint, $\boldsymbol{\lambda}_{mid}$, which is deemed "adjustable", is chosen halfway between $\boldsymbol{\lambda}_{start}$ and $\boldsymbol{\lambda}_{goal}$ and 5.0cm above the table to avoid dragging the cube. The waypoint, $\boldsymbol{\lambda}_{mid}$, and its time, t_{mid}, are selected as the initial policy parameters,

$$\boldsymbol{\theta}_{i=0} = \begin{bmatrix}\boldsymbol{\lambda}_{mid} \\ t_{mid}\end{bmatrix}. \tag{9.11}$$

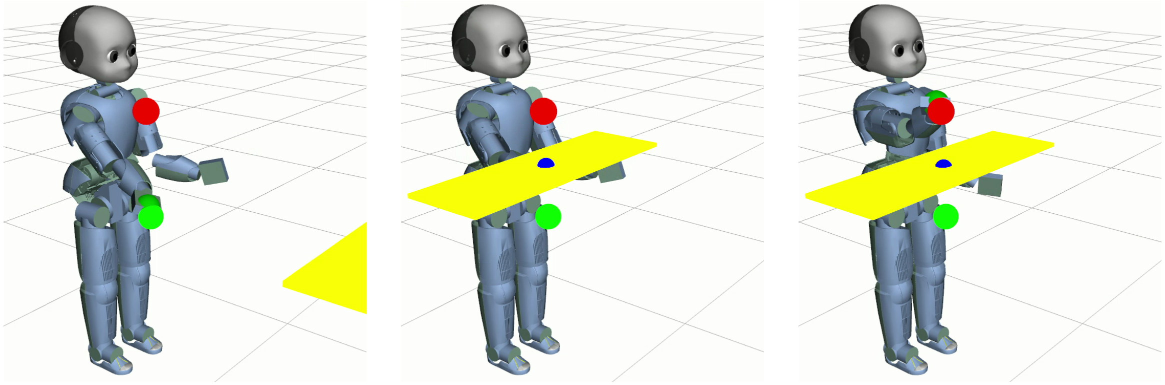
Therefore, $\boldsymbol{\theta} \in \mathbb{R}^4$. In this simulation, the optimization bounds are restricted to a hyperrectangle centered at $\boldsymbol{\theta}_{i=0}$. The bounds are given by,

$$\boldsymbol{\theta}_{i=0} \pm \begin{bmatrix} 0.05 \\ 0.05 \\ 0.02 \\ 1.0 \end{bmatrix} . \tag{9.12}$$

We restrict the search space in this example to try to prevent the robot from exploring movements which would cause it to fall. Task execution is terminated if the cube has reached its goal location to within 3.0cm or the movement has exceeded $t^{\max}_{\boldsymbol{\pi}} = 15.0$s in total duration. Qualitatively, the principle sources of task incompatibility and infeasibility are the heavy payload, which the actuators of the right arm are incapable of correctly supporting, and the incompatibility between the right hand task and the overall balance of the robot.

9.4 Results: TCFM using GPTs and BO

This section presents the results for the test simulations described in Sec. 9.3. For both tests, the time of an optimization iteration is equal to the time it takes to complete an execution, which is limited in both simulations to 10s and 15s respectively, plus the time it takes to compute the BO update, which never exceeds 1.0s. The BO update occurs in parallel to the task executions and takes less time to compute than the simulation does to reset for another rollout.

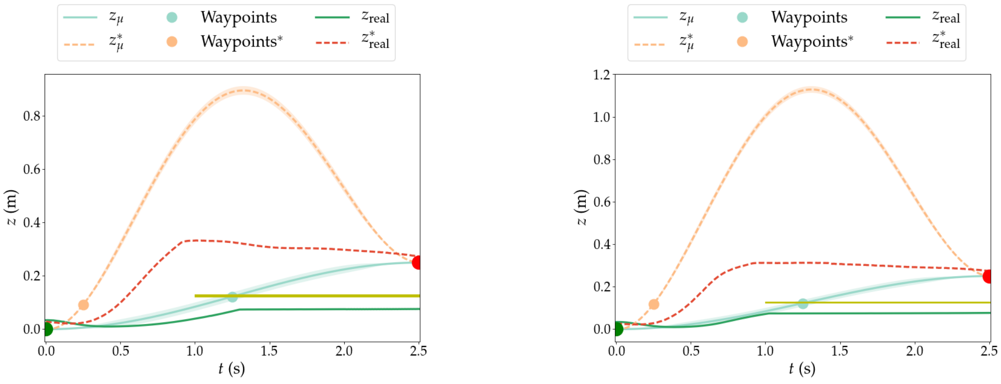


(a) starting position (b) original task (c) optimized task

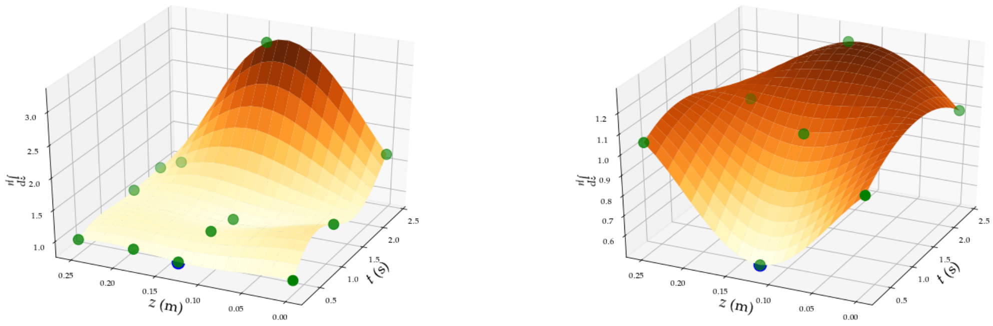
Figure 9.4 – This figure shows the original and optimized policies for the "Operational-Space Infeasibilities" test case detailed in Sec. 9.3.1. (a) shows the initial starting posture. In (b) it can be seen that the robot is blocked by the yellow obstacle introduced at 1.0s. TCFM generates a trajectory which causes greater acceleration at the beginning of the movement, moving the right arm above the obstacle before it arrives (c). This allows the robot to reach its goal, indicated by the red sphere.

9.4.1 Test Case 1: Operational-Space Infeasibilities

In Fig. 9.4, the main results of the simulation are presented. Figures 9.4b and 9.5b show that the original task execution fails due to the yellow obstacle impeding the movement of the hand. Applying TCFM to this problem both with and without stochastic policies results in an optimized right hand task which achieves its goal position and avoids the obstacle as shown in Fig. 9.4c. The solutions found using static and variable weights are identical as can be seen in Fig. 9.5a and Fig. 9.5b, but TCFM obtains the solution in only 7 rollouts using variable weights, while it takes 12 rollouts using static weights. This can



(a) static (b) variable



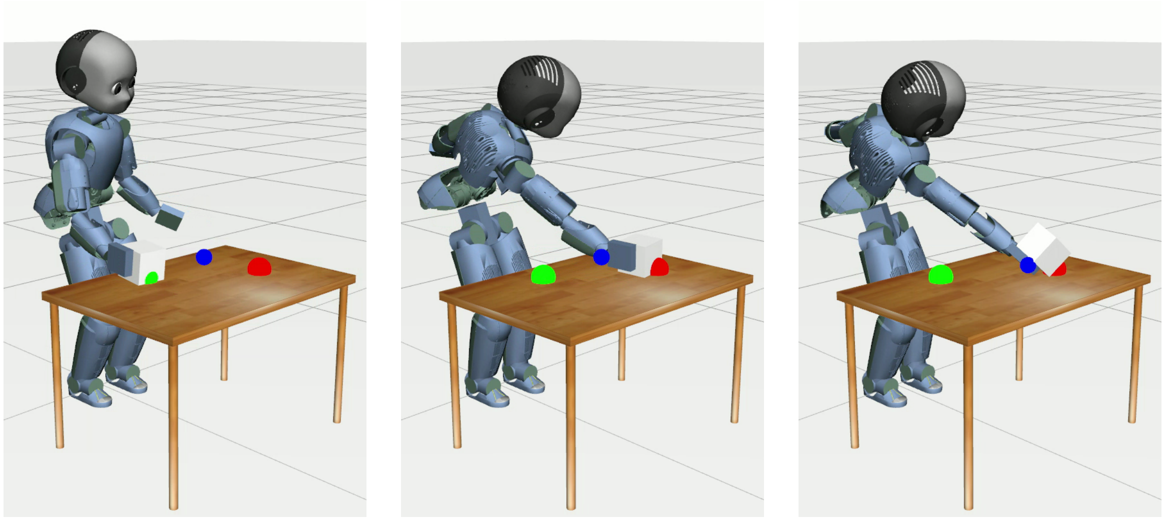
(c) static (d) variable

Figure 9.5 – (a) and (b) Plots of the z component of the original and optimized hand trajectories, using static and variable weights respectively. The notation $\bullet^*$ indicates an optimal value, or a value associated with the optimized policy. (c) and (d) a plot of the surrogate function cost mean, $\hat{j}^P_\mu$, of the BO update strategy within the TCFM, using static and variable weights respectively. These figures show the learned performance cost landscape of the obstacle avoidance simulation, after convergence of TCFM. The task rollouts, or cost function samples, are plotted by the green dots. The blue dot represents the optimal variables found for this simulation. Using these values for the task's optimization waypoint, the obstacle is successfully avoided, i.e. the effects of the task incompatibilities and infeasibilities are removed.

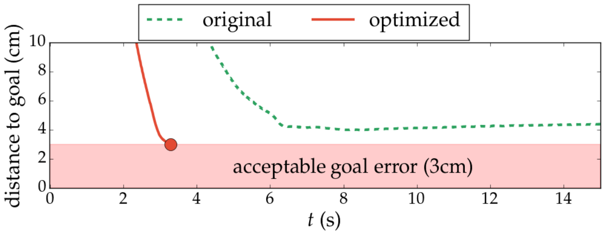
be confirmed in Fig. 9.5c and Fig. 9.5d where the BO surrogate function means, $\hat{j}^P_\mu$, are plotted for the static and variable optimizations. Here, we observe that the cost landscape generated by this disturbance scenario is most elevated after the obstacle insertion time and below the z-height of the obstacle. This is logical because any waypoint in this region of the search space would result in a failure to reach the goal. Each sample of the real cost function is indicated by a green dot, and represents a task execution. The blue dot shows the optimal value found by BO with high confidence and corresponds to the optimal waypoint shown in Fig. 9.5c and Fig. 9.5b.

The optimized right hand task trajectories shown in Fig. 9.5a and Fig.9.5b move the hand quickly enough to avoid the obstacle and attain its goal. One of the key things to note is that the optimal waypoint found for this task actually generates a trajectory outside of the robot's workspace. This has the effect of creating strong accelerations in the beginning of the movement allowing the robot to avoid the obstacle. Normally such a trajectory would generate dangerous movements, but the whole-body controller does not execute the infeasible portions of the trajectory because they violate the control constraints and are incompatible with the other tasks. This can be seen in Fig. 9.5a and Fig.9.5b where the real task trajectory, z^*_{real}, only partially follows the reference, z^*_{μ}. This is interesting because the optimized trajectory is more counterintuitive than the original trajectory, yet it is an effective solution to this problem. When stochastic policies are used, the tracking of the right hand task trajectory is somewhat smoother than that of the deterministic policy tracking. This occurs because the weight of the right hand task diminishes between the middle and goal waypoints and allows the controller in the stochastic policy to diminish the high desired accelerations of the task. This phenomenon allows the controller to “filter” overly aggressive trajectories which may be close to optimal but ultimately fail. When static weights are used, the controller attempts to track the aggressive desired accelerations and this leads to unstable behaviors which fail to reach the goal point, thus applying a high cost to nearly optimal policy parameters. This can be seen in the relatively flat cost landscape around the optimal parameters in Fig. 9.5c.

9.4.2 Test Case 2: Moving a Heavy Weight



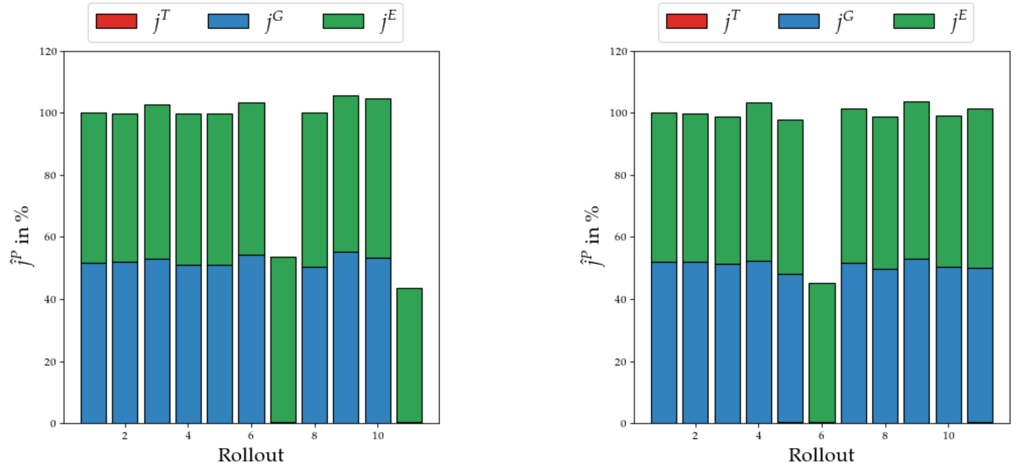
(a) starting position (b) original task (c) optimized task



(d) distance to goal

Figure 9.6 – This figure shows the original and optimized tasks for the “Moving a Heavy Weight" simulation detailed in Sec. 9.3.2. (a) shows the initial starting posture. In (c), it can be seen that the robot comes up just short of its goal location. By optimizing the middle waypoint of the task, the robot is able to accelerate more at the beginning of the movement, building up sufficient inertia to reach its goal within the acceptable error, as demonstrated by (d).

In Fig. 9.6, the original and optimized task executions for this example are shown. Although the differences are imperceptible in these still frames, the robot manages to attain the desired weight target to within 3.0cm of error, as shown by Fig. 9.6d. As with the first test case, the optimal behavior found with and without stochastic policies is the same. However, this time the TCFM finds an optimal set of policy parameters in the same number of rollouts: 11 in this test case. After 11 rollouts, the robot learns how to accomplish its objectives by placing the middle waypoint closer to the goal and sooner in the trajectory resulting in a task trajectory with higher initial accelerations. These accelerations increase the task error term and consequently the optimal torques needed to accomplish it. This causes the whole-body controller to preferentially accomplish the hand task and deviate the CoM task. Task equilibrium is reestablished at the end of the hand movement when the velocities and accelerations go to zero. This solution is quite intuitive to anyone who has moved a heavy object; however, it would be difficult to plan such a dynamic movement, without incorporating complex and contextually specific model information into the planning algorithm.



(a) static

(b) variable

Figure 9.7 – Evolution of the performance cost in percentage using static, (a), and variable, (b) weights (deterministic and stochastic policies). Each rollout cost is broken up into the component tracking, j^{T}, goal, j^{G}, and energy, j^{E}, costs.

Looking at Fig. 9.7, the cost evolution for the TCFM using static and variable weights it presented. The scaled performance cost $\hat{j}^{\mathrm{P}}$ (see Sec. 5.6.2) for each rollout is presented as a percentage,

$$\hat{j}_r^{\mathrm{P}} = \frac{j_r^{\mathrm{P}}}{j_0^{\mathrm{P}}} \times 100, \tag{9.13}$$

to facilitate the analysis of the evolution. Firstly, it can be seen that the goal and energy costs dominate, the scaled performance cost, $\hat{j}^{\mathrm{P}}$, for this particular scenario, which is due to the explosion of these terms when the robot falls and becomes unstable. The successful behaviors can be seen to have nearly zero goal cost and because virtually every other tested set of parameters results in a fall, the BO update strategy converges quickly to the same optimal parameters using static and variable weights.

9.4.3 Conclusions: TCFM using GPTs and BO

These experiments demonstrate that by using a more compact policy parameterization, GPTs, and a more efficient update strategy, BO, TCFM is able to account and compensate for complex task incompatibilities and infeasibilities in very few rollouts. Being efficient in the number of rollouts needed to perform TCFM is crucial to its applicability because it means it can be implemented on real-world systems. This is key to the utility of TCFM because a large part of task incompatibility and infeasibility stems from un-modeled perturbations which occur only in the real world. As such, this implementation of TCFM could be used to measure and correct these effects. Here, we use contrived scenarios to simulate such un-modeled perturbations or modeling errors, but they are a proof of concept of the capabilities of the method.

In Fig. 9.8, we show the number of iterations necessary to optimize the incompatible and infeasible right hand tasks with static and variable weights. In the obstacle avoidance scenario, we see a marked gain in convergence speed when using variance modulated weighting. On the contrary we see no difference for the scenario where the iCub must move a heavy load. These results can be attributed to the search bounds used for the two examples (see (9.8) and (9.12)).

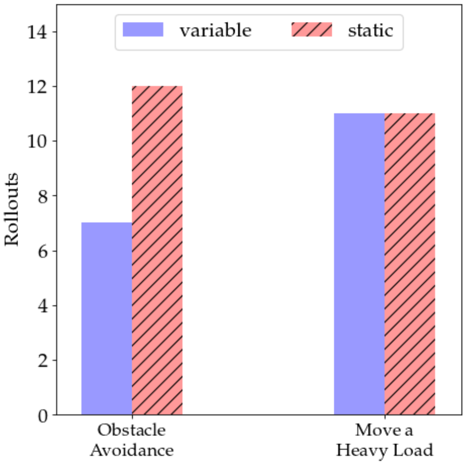


Figure 9.8 – A comparison of the TCFM with deterministic and stochastic policies (static and variable weights).

In the simple obstacle avoidance case, the search bounds allow for movements which generate high accelerations. This is primarily due to the temporal dimension of the search. When two waypoints are temporally close, but spatially distant, the resulting trajectory will produce large accelerations. As such, during the variance modulated weight trials, the trajectories which cause the posture tasks to deviate wildly, are partially suppressed during periods of high variance. This effectively ignores the useless components of the trajectory while benefiting from those which improve its compatibility cost. In the static weight case, the trajectories are followed strictly and this produces highly energetic and dangerous movements in many cases. For second simulation scenario, the optimization bounds are far more restricted in order to prevent dangerous exploration, which essentially nullifies the benefits of variance modulated weighting. From the two test scenarios, using stochastic policies instead of deterministic policies, seems to have a beneficial effect on the sample efficiency of TCFM. However, this exploration does not provide enough data to draw any statistically significant conclusions about whether or not stochastic policies truly improve sample efficiency. Also, without experimenting on a real robot, there is no way of knowing if stochastic policies are even stable. These matters require further testing and development, because as shown with the test cases from Sec. 9.1, GPTs seem very promising for reactively improving task compatibility and feasibility.

One downside to GPTs is that you must specify the time of the waypoints, which is not always easy, and it can generate trajectories which actually violate the BO bounds, as shown by Fig. 9.5b. While this can have some interesting benefits, for the most part when bounds are specified, it is smart to respect them. Furthermore, because BO uses GPs to model the latent cost function, the size of the surrogate grows rapidly with the number of parameters and rollouts. A naive[1] implementation of BO suffers from the curse of dimensionality, and beyond ten optimization variables, modern computers quickly run out of memory. Therefore, what remains to be seen is:

1. Can a simpler more robust policy parameterization be used, which generates trajectories within bounding constraints?

---

[1] The most basic implementation of BO requires that the search space be discretized, which causes the memory requirements to grow exponentially due to the curse of dimensionality. This makes BO a global optimization method but is not effective for problems with more than ten dimensions. Most modern implementations use local stochastic or gradient update strategies seeded throughout the search space to more efficiently handle large problems. These remove any assurance of global optimality but pay off in terms of practicality.

2. How well do less efficient update strategies like CMA-ES perform with these low-dimensional policy parameterizations, and how does it compare with BO?

Chapter 10

Robust and Sample Efficient TCFM for Real Robots

In Chapter 9, it is shown that TCFM can efficiently optimize whole-body behaviors by using Gaussian Process Trajectories (GPTs) for the policy parameterization as well as Bayesian Optimization (BO) as the update strategy. While this is a first step towards real world implementation, GPTs require that the trajectory waypoint times be specified a priori. While not overly constraining, this is practically difficult for many tasks where acceleration and velocity constraints need to be respected. Furthermore, GPTs may not respect the optimization bounds of TCFM. In an effort to determine a more compact and robust policy parameterization, Time-Optimal Trajectories (TOT) are used here for task trajectory generation. TOTs require only waypoints and maximum velocity and acceleration limits to generate a time-optimal trajectory. Additionally, these TOTs generate paths which go from waypoint to waypoint in a straight line and blend those segments with circles, ensuring that the path is bounded by the waypoints, thus respecting the TCFM bounds. Because the parameterization is compact, it remains to be determined if stochastic optimization update strategies can efficiently optimize the parameters. Therefore, a comparison of TCFM using CMA-ES and BO as update strategies is made. This is important because stochastic optimization update strategies are generally more efficient than BO update strategies in high dimensional problems, and if TCFM is to be extended to optimize many tasks in the same behavior, then the parameter space will grow beyond what BO can effectively handle. The robustness of TCFM is tested on 100 random reaching trials, and the power of TCFM is demonstrated on the complex sitting to standing trials. TCFM is shown to successfully generate policies which reach previously unreachable targets, and stand up without losing balance. The genericity of TCFM is demonstrated by changing the underlying whole-body control hierarchy in the policy. In both simulation and on a real iCub, TCFM is shown to efficiently and robustly maximize task compatibility and feasibility.

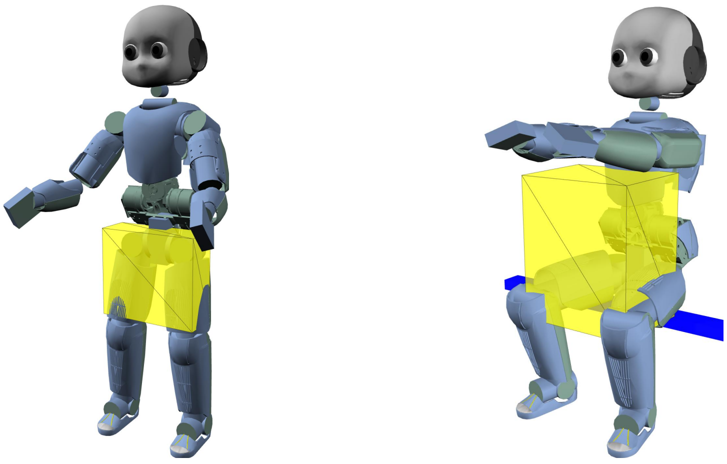
10.1 Experimental Setup

.

In the following test cases, we look at the impact of CoM tasks on the performance of the whole-body behaviors. CoM tasks are crucial to humanoid robotics because they allow balance to be achieved both statically and dynamically. They are commonly the most important task in a set and also the reason why other tasks may never be completed. Throughout the following examples, only the CoM task trajectory is optimized in TCFM. The bounds applied in TCFM ensure that the CoM position respects static stability constraints, i.e. its ground projection stays within the PoS. This precludes the need for applying similar whole-body controller inequality constraints on the CoM position.

The first test case explores basic reaching movements under bipedal equilibrium, and serves as a benchmark for TCFM with TOTs and CMA-ES/BO. It provides useful statistics to analyze the method and the CMA-ES and BO update strategies used in the TCFM. The rest of the test cases present a more dynamically complex scenario in which the robot starts from a seated position and must transition to standing. Here, we look at the difficulties of contact transitioning and dynamic equilibrium. In both experiments, balance is achieved by keeping the robot's CoM position over its PoS.

For both the reaching and standing scenarios, the CoM task trajectory parameters are optimized to improve the performance cost, j^{P}. Box constraints for the optimization can be applied in both BO and CMA-ES and are set here using static stability constraints, i.e. the projection of the CoM must remain inside the PoS. The z bounds are chosen as $0.3\text{m} \leq z \leq 0.52\text{m}$, which are the heights of the CoM when the robot is squatting and when it is upright with its arms stretched upwards, respectively. The bounding boxes for both experiments are shown in Figs. 10.1a and 10.1b. In all of the test cases presented here, Gazebo is used as the simulation environment with the ODE physics engine.



(a) reaching

(b) standing

Figure 10.1 – Figures (a) and (b) show the policy parameter, $\boldsymbol{\theta}$, bounding boxes used by the TCFM update strategies for the reaching and standing scenarios, respectively.

10.1.1 Test Case 1: Reaching

The task set and constraints used in this test case are presented in Tables 10.1 and 10.2. The head orientation task, which is used in the previous experiments, is removed here because it is redundant with the postural task for all the DoF. The objective of this test case it to demonstrate the flexibility of the proposed TCFM, as well as to glean useful statistics about the two proposed update strategies, BO and CMA-ES.

Table 10.1 – Task set for test case 1 of TCFM with TOTs and CMA-ES/BO. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Joint Position	All DoF	25	10^{-5}	1
Cartesian	Torso	3	0.05	1.0
Cartesian	Right Hand	3	0.001	40
Cartesian	Left Hand	3	0.0001	40
Cartesian	CoM	3	1.0	50.0

Table 10.2 – Constraints for test case 1 of TCFM with TOTs and CMA-ES/BO.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×4
Contact Set	Right Foot	6×4

To accomplish this, 100 reach targets are randomly generated around the simulated robot, see Fig. 10.2. For each target, a straight line trajectory is generated for the right hand task between its starting position and the position of the target. The CoM task trajectory is generated between two waypoints,

$$\Lambda_{\text{CoM}} = \begin{bmatrix}\lambda_{\text{start}} & \lambda_{\text{goal}}\end{bmatrix}, \tag{10.1}$$

where λ_{start} is the initial CoM position, and λ_{goal} is the desired goal CoM position, which is initially chosen to be the center of the PoS at the current CoM z height — i.e. to stand still. In this test case, the goal waypoint, λ_{goal}, is used as the policy parameters which is optimized by TCFM.

$$\boldsymbol{\theta}_{i=0} = \begin{bmatrix}\lambda_{\text{goal}}\end{bmatrix}. \tag{10.2}$$

The cost function is again the performance cost, j^{P},

$$j_i = \text{cost}\left(\{\mathcal{S}, \mathcal{A}\}_i\right) = j^{\text{P}}, \tag{10.3}$$

and is computed using only the right hand and CoM tasks,

$$j^{\text{P}} = \frac{\left(j^{\text{TP}}_{\text{right}} + j^{\text{TP}}_{\text{CoM}}\right) + \varphi j^{\text{E}}}{t^{\text{end}}_{\boldsymbol{\pi}}}, \tag{10.4}$$

and $\varphi = \max\left(\|\boldsymbol{\tau}(k)\|_2\right)^{-1}$, the maximum torque norm from the original policy.

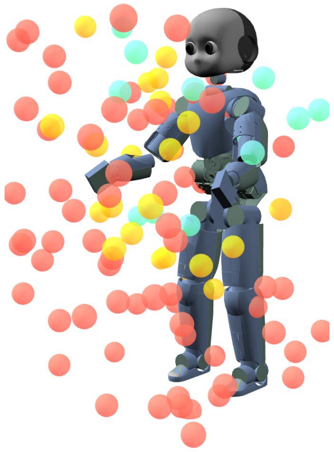


Figure 10.2 – Random reaching targets used to provide a statistical analysis of TCFM. The target spheres are color coded to indicate their test case, with green meaning reachable, orange meaning possibly reachable, and red meaning unreachable.

The objective of the experiment is to attain the reach target and a target is considered attained when the right hand task frame is within 3.0cm of it. The rollout is stopped if the target is attained or if a time limit, $t_{\boldsymbol{\pi}}^{\max}$, is exceeded. For each reach target, the TCFM is performed using both BO and CMA-ES as update strategies. The target is then classified into one of three cases. If the robot attains the target with the original policy, the target is considered **reachable**. If it is unable to attain the target with the original policy, but attains the target with the optimized policy, then the target is **possibly reachable**. Finally, if the reach target is unattainable with either the original or optimized policies, then it is considered **unreachable**. Using one of the results from the possibly reachable cases, the original and optimized policies are executed on the real iCub to see if the behaviors produced by TCFM are viable on real systems.

10.1.2 Test Case 2: Standing

Table 10.3 – Task set for test case 2 of TCFM with TOTs and CMA-ES/BO. $K_d = 2\sqrt{K_p}$ for all tasks.

Task Type	**DoF/Link**	**Dimension**	**Weight w**	**Gain K_p**
Joint Position	All DoF	25	10^{-4}	5
Orientation	Torso	3	0.002	10.0
Cartesian	Right Hand	3	0.001	20.0
Cartesian	Left Hand	3	0.001	20.0
Cartesian	CoM	3	1.0	60.0

Table 10.4 – Constraints for test case 2 of TCFM with TOTs and CMA-ES/BO.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×4
Contact Set	Right Foot	6×4
Contact Set	Left Upper Leg	6×2
Contact Set	Right Upper Leg	6×2

In this experiment, the robot is seated on a stationary bench and the objective is to stand up. This simple, yet dynamically complex whole-body behavior presents a significant challenge for TCFM because the difference between standing and falling, success or failure, is exceedingly small. To get the robot to stand up from a seated position, the following task set and constraints, presented in Tables 10.3 and 10.4, are used. The bench contacts, at the Left and Right Upper Legs, are 22cm from the ground and on the back of the iCub's upper thigh links. In this case the only primary task is the CoM task, and its TOT is parameterized by three waypoints,

$$\Lambda_{\text{CoM}} = \begin{bmatrix} \lambda_{\text{start}} & \lambda_{\text{middle}} & \lambda_{\text{goal}} \end{bmatrix}, \tag{10.5}$$

where λ_{start} and λ_{goal} have the same meaning as in (10.1) and λ_{middle} is a middle waypoint between the start and goal CoM positions. Here, λ_{goal} is picked as a nominal standing CoM position near the middle of the PoS in x and y and at approximately 0.5m from the ground, and λ_{middle} is picked as a point halfway between λ_{start} and λ_{goal}. In this test case, the middle waypoint, λ_{middle}, is used as the policy parameters in the TCFM,

$$\boldsymbol{\theta}_{i=0} = \left[\lambda_{\text{middle}}\right]. \tag{10.6}$$

The performance cost, j^{P}, is computed using only the CoM task,

$$j^{\text{P}} = j^{\text{E}} + j^{\text{TP}}_{\text{CoM}}. \tag{10.7}$$

Both CMA-ES and BO are used as update strategies within the TCFM for this test case.

In order to stand, the Left and Right Upper Leg contacts used in the whole-body controller must be deactivated or the robot will never be able to get up. Thus, the bench contacts are deactivated arbitrarily at 2.0 seconds. The rollout is executed until the CoM task has attained λ_{goal} to within 3.0cm of error or some time limit, $t^{\max}_{\boldsymbol{\pi}}$, has been exceeded.

10.1.3 Test Case 3: Standing with Help

Using the same experimental setup described by test case 2, an external support is simulated, in a simple and naive way, as an external force applied at each forearm of the robot throughout the overall movement. This supporting force is designed to mimic a person helping the robot to stand up, which can be regarded as a dynamic perturbation when not accounted for in the whole-body controller model. The magnitude of the force at each forearm is chosen small: $F_{support} = 5.23\ N$ pointing 45° upward and front in the sagittal plane of the robot. This force corresponds to the minimum required assistive force for the original policy to produce a successful standing motion. This experiment is designed to show how TCFM responds to changes in the underlying dynamics of the policies. Only BO is employed as a update strategy in this test case.

10.1.4 Test Case 4: Standing with a Different Whole-Body Controller

The objective here is to determine if TCFM is truly generic and controller agnostic. To accomplish this, the whole-body controller within the policy is changed. The whole-body controller used in this test case is a momentum-based hierarchical controller developed in [Pucci et al., 2016; Nava et al., 2016], which has momentum tracking and joint impedance tasks — the most important of which is the former. For the momentum task, the desired value is entirely determined by the desired CoM acceleration, $\ddot{\boldsymbol{x}}^{\text{des}}_{\text{CoM}}$, and is provided by a feedforward Proportional-Integral (PI) servoing controller,

$$\ddot{\boldsymbol{x}}^{\text{des}}_{\text{CoM}} = \ddot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}} - K_p(\dot{\boldsymbol{x}}_{\text{CoM}} - \dot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}) - K_i \int_{t=0}^{t} (\boldsymbol{x}_{\text{CoM}} - \boldsymbol{x}^{\text{ref}}_{\text{CoM}})dt, \tag{10.8}$$

where K_p and K_i are the proportional and integral gain matrices respectively. The CoM reference values, $\boldsymbol{x}^{\text{ref}}_{\text{CoM}}$, $\dot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}$, and $\ddot{\boldsymbol{x}}^{\text{ref}}_{\text{CoM}}$ are provided by the CoM TOT. The choice of a feedforward PI servoing controller is imposed by the new control architecture.

The task set is then composed as shown by Table 10.5 and the constraints are shown in Table 10.6.

Table 10.5 – Task set for test case 4 of TCFM with TOTs and CMA-ES/BO. Levels are indicated in ascending order of importance.

Task Type	**DoF/Link**	**Dimension**	**Level**	**Gain K_p/K_d**
Joint Impedance	All DoF	25	0	9/6
Momentum	CoM	3	1	20.0/ -

Table 10.6 – Constraints for test case 4 of TCFM with TOTs and CMA-ES/BO.

Constraint Type	**DoF/Link**	**Dimension**
Joint Limit	All DoF	25×2
Torque Limit	All DoF	25×2
Contact Set	Left Foot	6×2
Contact Set	Right Foot	6×2
Contact Set	Left Upper Leg	6×2
Contact Set	Right Upper Leg	6×2

For this whole-body control architecture, a finite-state-machine (FSM) composed of two states, coordinates the standing motion in the controller. The initial posture is chosen to ensure that the ground-plane (x-y) projection of starting CoM position is within the PoS defined by the bench and ground contact locations. In the "Sit" state, the robot is seated on the bench, and the two contacts at the left and right upper legs are controlled to keep the equilibrium. The toes are in contact with the ground. When a resultant ground reaction force greater than 150N is detected, the FSM switches to the "Stand" state, moving the bench contacts to the left and right heels in the whole-body controller.

As with test case 2 from Sec. 10.2.2 the middle waypoint of the CoM TOT is used as the policy parameters,

$$\boldsymbol{\theta}_{i=0} = \left[\lambda_{\text{middle}}\right], \tag{10.9}$$

and the performance cost, j^{P}, is computed using only the CoM task,

$$j^{\text{P}} = j^{\text{E}} + j^{\text{TP}}_{\text{CoM}}. \tag{10.10}$$

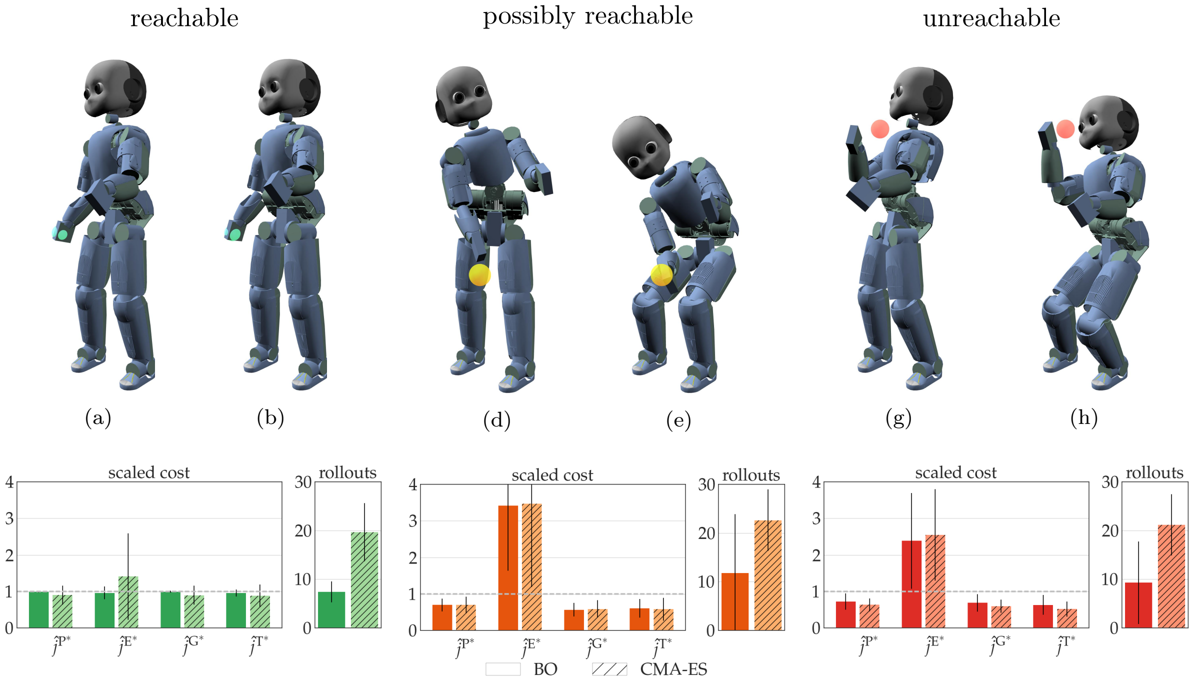
The rollouts are first carried out in simulation using Gazebo as the simulation environment with the ODE physics engine.

Bootstrapped and Non-Bootstrapped Rollouts on the Real iCub

TCFM is iterated until the convergence criterion detailed in Sec. 5.7.2 is satisfied or a maximum number of rollouts is exceeded. In this study, $\gamma = 1.0$cm, (see (5.31)) and the maximum number of rollouts is 30 in simulation and 10 on the real robot. The optimal policy parameters, $\boldsymbol{\theta}^*$, are then used to generate $\boldsymbol{\pi}^*_{\boldsymbol{\theta}}$ which is rolled out on the real iCub. This rollout is used to demonstrate that TCFM can be performed in simulation and produce compatible and feasible behaviors on the real robot.

With the $\boldsymbol{\pi}^*_{\boldsymbol{\theta}}$ from simulation as a starting point, the TCFM is continued by performing rollouts on the real iCub. For these rollouts we look at two cases. In the first, the BO surrogate function training is *bootstrapped* with training data from the simulated rollouts and further trained on data from the real rollouts. In the second *non-bootstrapped* case, the surrogate function is trained only using the real rollout data. For both cases, the $\boldsymbol{\pi}^*_{\boldsymbol{\theta}}$ from the simulation rollouts is used as the initial policy for the real rollouts, warm-starting the TCFM. To limit the number of falls on the real robot, the BO search space bounds are restricted to a 10cm cube around the initial $\boldsymbol{\theta}^*$, for the real rollouts. Ten rollouts are performed for both cases. Only BO is employed as a update strategy in this test case.

10.2 Results



(a) (b) (d) (e) (g) (h)

(c) (f) (i)

Figure 10.3 – Results of 100 reaching experiments used to study the task compatibility optimization method. An average example is presented for the three possible reach cases. In the reachable case, (a) & (b), both the original and optimized policies attain the reach target. In the possibly reachable case, (d) & (e), the original policy does not attain the target but the optimized policy does. Finally in the unreachable case, (g) & (h), neither policy attains the target, but the optimized policy reduces the target error. For each case, the scaled cost means and standard deviations are plotted for both the BO and CMA-ES update strategies. Any scaled cost lower than 1.0 is an improvement (the 1.0 line is indicated by a dashed grey lines). For all three cases the scaled performance cost $\hat{j}^{P^*}$ is always less than or equal to 1.0. This indicates that the optimized policies will always be as good if not better than the original policies. In the reachable case, (c), little improvement is seen because the tasks are already compatible. For the possibly reachable and unreachable cases, (f) and (i), the compatibility is improved by reducing the tracking and goal costs at the expense of increased energy usage. For each of the cases, BO outperformed CMA-ES in number of rollouts for convergence on average, but was less consistent across-the-board.

In this section, the results for the reaching and standing experiments are presented. To better understand the scenarios, results and analyses, it is strongly recommended to watch the video.

10.2.1 Test Case 1: Reaching

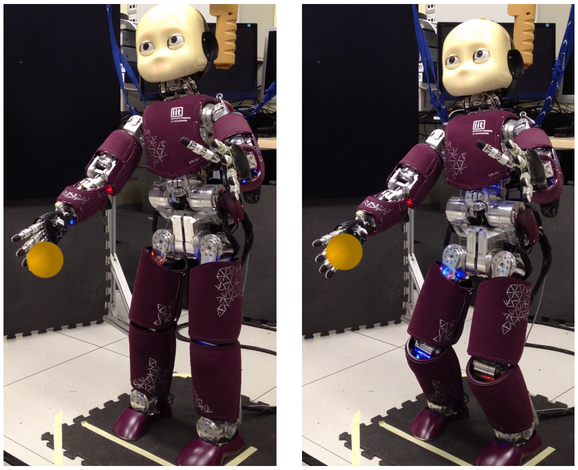

(a) original (b) optimized

Figure 10.4 – Original and optimized reaching behaviors executed on an iCub robot. These preliminary results show that the behaviors produced by TCFM are viable on real platforms.

In Fig. 10.3, representative examples of the three reach cases are shown. For each, the scaled cost means and optimization iteration means are computed. The scaled performance cost, $\hat{j}^{\mathrm{P}^*}$, and the component scaled costs, $\hat{j}^{\mathrm{E}^*}$, $\hat{j}^{\mathrm{G}^*}$, and $\hat{j}^{\mathrm{T}^*}$, are calculated by dividing the optimized costs by the original costs. Rather than presenting the scaled task performance cost, $\hat{j}^{\mathrm{TP}^*}$, the scale component goal and tracking costs are broken out,

$$j^{\mathrm{T}^*} = j^{\mathrm{T}^*}_{\mathrm{right}} + j^{\mathrm{T}^*}_{\mathrm{CoM}} \tag{10.11}$$

$$j^{\mathrm{G}^*} = j^{\mathrm{G}^*}_{\mathrm{right}} + j^{\mathrm{G}^*}_{\mathrm{CoM}}, \tag{10.12}$$

to better analyze their effects on TCFM.

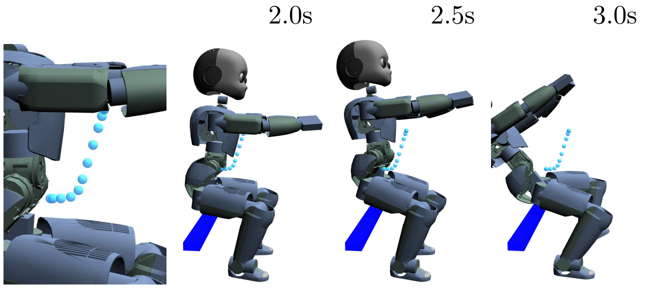
In the reachable case, Figs. 10.3a, 10.3b and 10.3c, we can see that the original task trajectories go unmodified because they are already compatible and feasible. In a few cases, CMA-ES is able to improve slightly on the performance cost, by finding faster ways to reach the same target, which can be interpreted by the decreased goal and tracking costs and increased energy costs. We start to see performance cost improvements between 30%-50% in the possibly reachable case, Figs. 10.3d, 10.3e and 10.3f, where both update strategies quickly converge on solutions which reduce the tracking and goal errors, allowing the robot to attain the target. These improvements require increased energy usage to move the CoM, but the resulting successful reach finishes more quickly, amortizing the impact of the increased energy cost. In the unreachable case, Figs. 10.3g, 10.3h and 10.3i, we see cost improvements similar to those in the possibly reachable case, even though the targets are physically unattainable.

For all three cases, BO and CMA-ES show similar performance in terms of cost reduction. In terms of convergence, BO tends to find an optimum in approximately half the number of rollouts needed by CMA-ES, however, it is less consistent, i.e. has higher variance in the results, than CMA-ES across all cases.

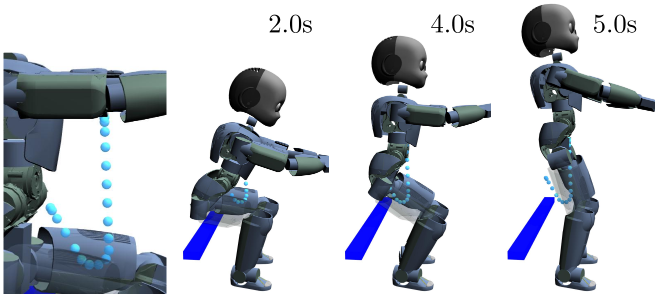
To validate if the whole-body behavior optimized by TCFM is viable on a real system, a possibly reachable case is selected at random and executed on the real iCub. In Fig. 10.4, the original, Fig. 10.4a, and optimized, Fig. 10.4b, policy rollouts are shown. The right hand reaching target is superimposed on the images. It can be seen that the optimized policy gets the hand inside of the target zone thanks to the bending of the knees caused by the optimized CoM task trajectory.

10.2.2 Test Case 2: Standing

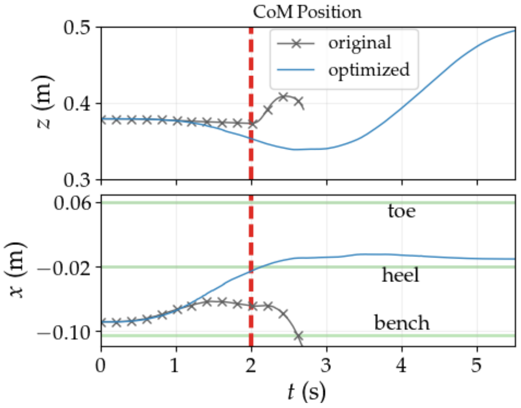
In Fig. 10.5, we see the evolution of the CoM for the original and optimized policies. The whole-body motion produced by the original policy, Fig. 10.5a, is unstable and causes the robot to loose balance. The optimized policy, on the other hand, produces a stable sit-to-stand transition as shown in Fig. 10.5b. In Fig. 10.5c we observe that, at the moment the bench contacts are deactivated in the controller (the dashed vertical red line), the original motion immediately tends to lift the CoM upwards, despite an inappropriate x-location of the CoM (not close enough to the foot PoS). This infeasible CoM trajectory does not respect the dynamic balancing conditions (see [Perrin et al., 2015]) and causes the robot to fall. A detailed analysis of the loss of balance is beyond the scope of this study. The optimized policy moves the CoM more aggressively in the forward direction as well as lowering it prior to the contact deactivation instant. The resulting CoM trajectory is balance consistent, thus leading to a successful sit-to-stand transition.



(a) original



(b) optimized

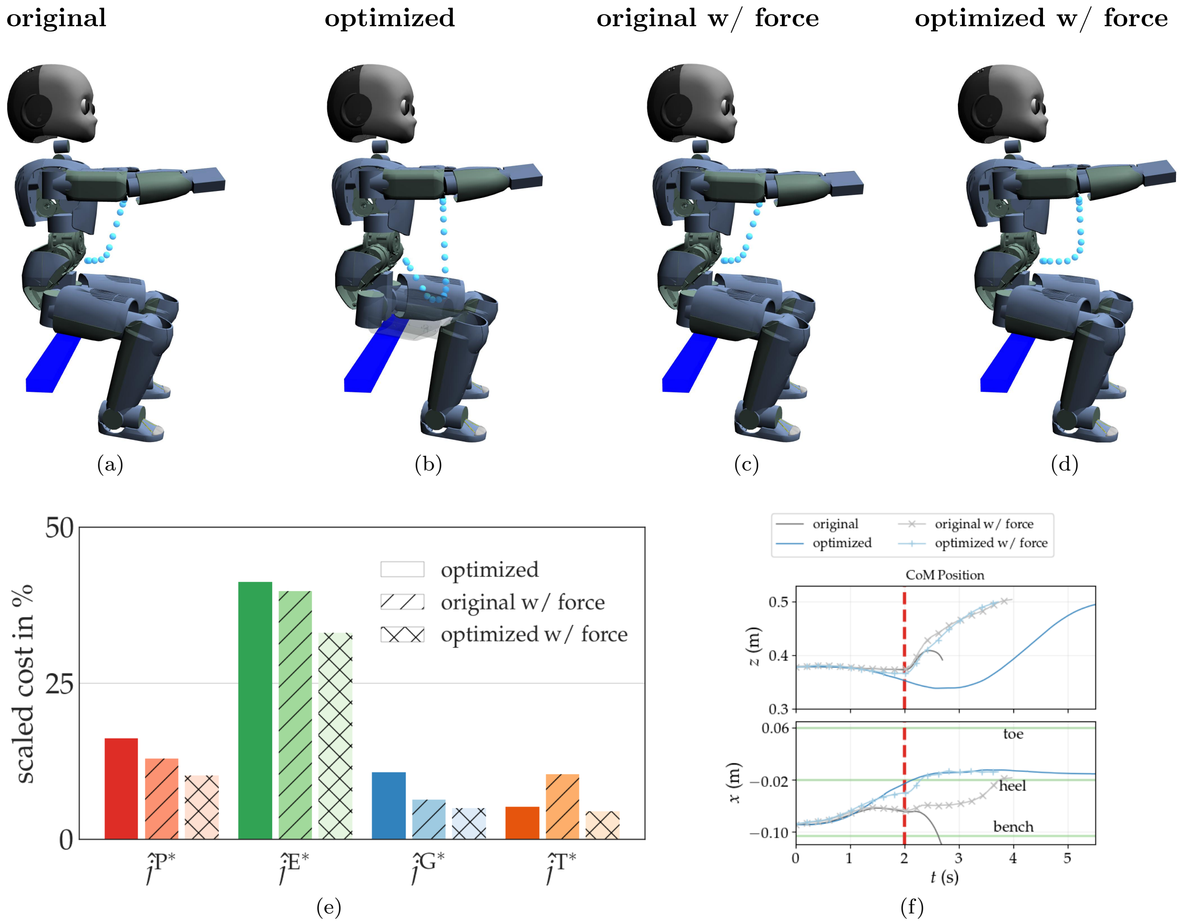


(c)

Figure 10.5 – Original and optimized CoM reference trajectories and their resultant whole-body motions. The original policy produces an unstable standing motion causing the robot to lose balance. The optimized policy, however, produces a successful sit-to-stand transition. The right hip is translucent in (b) to make the reference trajectory visible. (c) evolution of the CoM for the original and optimized policies. The original CoM curves are cut off after 2.7 seconds when the robot loses balance. The red dashed line indicates the moment when the bench contacts are deactivated in the whole-body controller.

10.2.3 Test Case 3: Standing with Help

The resulting CoM trajectories for the standing scenario with and without external support are presented in Fig. 10.6. The optimized, Fig. 10.6b, original with force, Fig. 10.6c, and optimized with force, Fig. 10.6d, policies all produce successful standing motions, but despite the original policy with force being successful, the TCFM still improves the performance cost by modifying the CoM trajectory to better exploit the supporting force. The results in terms of performance cost optimization are presented in Fig. 10.6e. As intuitively expected, the use of an external supporting force together with an optimized policy yields the best results in terms of tracking and whole-body energy expenditure. The main difference between the optimized policy without external support and the optimized policy with external support lies in the energetic expenditure and time needed to reach a standing posture. This is also illustrated in Fig. 10.6f where the time needed to reach the CoM height $z_{\text{CoM}} = 0.5\text{m}$ is more than 1.5 times larger (measured from the instant where bench contact constraints are no longer enforced) in the case where no external support is present.



(a) (b) (c) (d)

(e) (f)

Figure 10.6 – Original and optimized CoM reference trajectories with and without support. (a) The original policy produces an unstable standing behavior causing the robot to lose balance. The optimized without support, (b), original policy with support, (c), and externally supported, (d), cases produce successful sit-to-stand transitions. (e) The costs presented here corresponds to the three following cases: optimized policy without external support, original policy with external support and optimized policy with external support. Each cost is presented as a ratio of the original cost (no optimization, no external support). As intuitively expected, the use of an external supporting force together with an optimized policy yields the best results in terms of tracking and whole-body energy expenditure. (f) Evolution of the CoM for the original and optimized policies, with and without support. The original CoM curves without support are cut off after 2.7 seconds when the robot loses balance. The red dashed line indicates the moment when the bench contacts are deactivated in the whole-body controller.

10.2.4 Test Case 4: Standing with a Different Whole-Body Controller

In Fig. 10.7, we see the evolution of the CoM for the original policy, B 0, and the policies optimized in simulation, B 25, the bootstrapped case, B 33, and the non-bootstrapped case, NB 2. The bootstrapped rollouts are denoted, "B #", and the non-bootstrapped rollouts are denoted, "NB #". The initial straight line CoM trajectory produces an unstable policy, which causes the robot to loose balance. The failing (i.e. falling) rollouts are indicated by the hatched red backgrounds in Figs. 10.8a and 10.8b. Because the

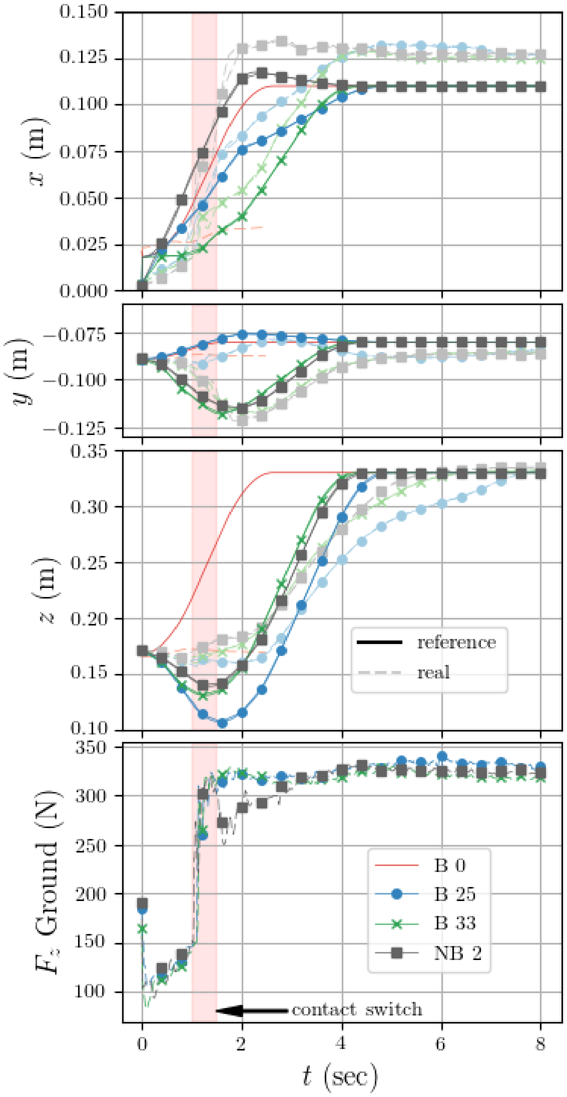


Figure 10.7 – Evolution of the CoM trajectories of the original and optimized policies. “B” indicates the bootstrapped case, and “NB” the non-bootstrapped case. B 0 is the original policy executed in simulation. The solid lines are the reference values generated by $\boldsymbol{\pi_\theta}$ and the lighter dashed lines are the real measured values. The original, B 0, real lines are cut off after 2.5s when the robot falls. The noisy B 0 force profile is omitted from the force plot, to not obfuscate the other force profiles.

initial policy fails, the measured CoM position values for B 0 are not shown after 2.5 seconds due to noise, and the F_z values are omitted completely for clarity. After 24 rollouts in simulation (see Fig. 10.8a), the TCFM converges to a policy which produces a successful sit-to-stand transition in both simulation and on the real robot. This policy comes from the rollout 21 in simulation, and is used as the policy for the initial real rollouts in both the bootstrapped and non-bootstrapped cases, B 25 and NB 0 respectively. This is confirmed by the real and reference CoM trajectories for B 25 in Fig. 10.7. Had the motion failed, the real values would not have tracked the reference values as is the case for B 0.

Looking at the z-axis and F_z plots in Fig. 10.7, we see that the optimal strategy, found in B 21, is to move the CoM downwards initially to increase the ground reaction force, and shift the robot’s weight to the feet. This shift must come early in the execution of the policy in order to achieve a contact switch in the FSM, and thus allow the CoM to continue tracking the trajectory references. When this policy is executed on the real robot in B 25 and NB 0, the results are successful, but higher $\hat{j}^{\mathrm{P}^*}$, than predicted by simulation, are observed for both cases. These discrepancies come as no surprise, but indicate that some unpredicted factors come into play on the real robot and must therefore be accounted for.

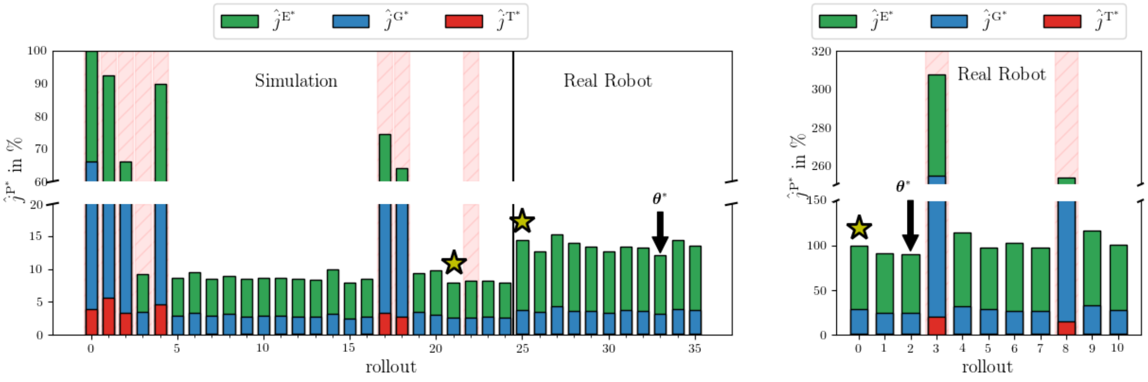
Looking at NB 2 and NB 3, we have an example of an optimal policy and a costly policy which produces a fall. In these two rollouts, the policy parameters being tested are,

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_2 = \begin{bmatrix} 0.12 \\ -0.124 \\ 0.115 \end{bmatrix}, \tag{10.13}$$

and

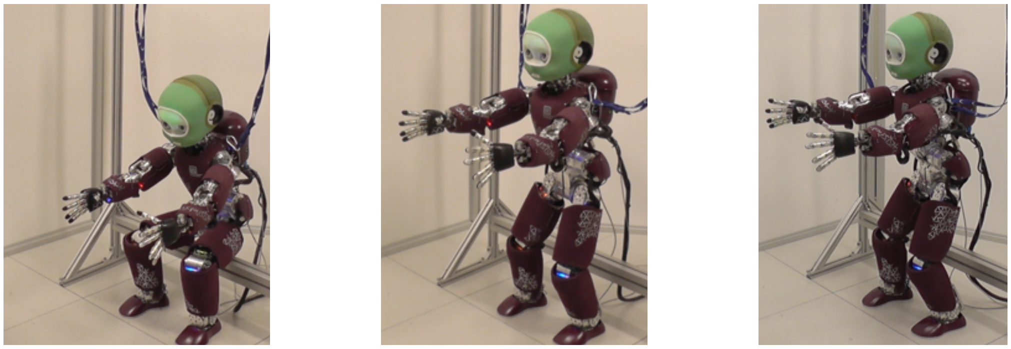
$$\boldsymbol{\theta}_3 = \begin{bmatrix} 0.12 \\ -0.02 \\ 0.115 \end{bmatrix}, \tag{10.14}$$

respectively. These parameters differ by only 10cm in the y-axis, which is not much, and in theory should not affect a sagittal plane motion. However, this subtle change in the trajectory makes the difference between optimality and catastrophic failure. We can see in the y-axis plot of Fig. 10.7 that the optimal policies found both with and without bootstrapping possess this y-axis motion, contrary to the policy optimized in simulation, and clearly attempt to compensate for un-modeled phenomenon present in the real system. Given the sensitive nature of the sit-to-stand motion, hand-tuning the trajectory parameters would be a difficult chore even for an expert.



(a) bootstrapped

(b) non-bootstrapped



(c) 0.0s

(d) 8.0s

(e) 6.0s

Figure 10.8 – (a) performance cost percentages (bootstrapped case) from the rollouts in both simulation and on the real robot. The rollouts which produced a failure (falling) are indicated by the red hatched backgrounds. The optimal (best observed costs) policy parameters, $\boldsymbol{\theta}^*$, are indicated for both real rollout cases. (b) costs for the non-bootstrapped case. The optimal policy found in the simulated rollouts comes from B 21 and is indicated by the yellow star. B 25 and NB 0, i.e the first real rollouts for the bootstapped and non-bootstrapped cases, use the B 21 policy and are also indicated by yellow stars. B 33 is the optimal policy found during the real bootstrapped rollouts. NB 2 is the optimal policy found during the real non-bootstrapped rollouts. (c) initial posture of the iCub robot. (d) and (e) final standing postures of the optimized motions for the bootstrapped and non-bootstrapped cases respectively.

Figures 10.8a and 10.8b show the component costs for each rollout with and without bootstrapping. The percentage improvement, $\hat{j}_i^{\mathrm{P}} \times 100$, of each cost shows how PS improves the motion with respect to the initial policy. The overall evolution of the total performance costs shows the almost binary nature of the sit-to-stand scenario — either the robot stands or it falls. Given this, and the nature of the BO used here, we do not observe smooth convergence. Furthermore, in both the bootstrapped and non-bootstrapped cases the convergence criterion from (5.31) is not attained. Nevertheless, the initial policies are improved using TCFM. The majority of this improvement is due to a decrease in energy consumption. The energy savings come primarily from the large sagittally actuated pitch joints, and most notably that of the torso pitch. In Fig. 10.9, we see the torques from B 25 and B 33. Both policies produce a successful sit-to-stand motion, but the optimized policy solicits this actuator less than the initial policy and reaps large gains in the energy cost. As expected, the rollouts without bootstrapping show more aggressive exploration, with two policy failures at NB 3 and NB 8, than the rollouts with bootstrapping. This comes from the higher variance associated with the un-explored regions of the policy parameter search space.

This aggressive exploration, however, leads to an optimized motion which moves faster from the starting seated posture (see Fig. 10.8c) to a standing posture, as shown by the trajectory in Fig. 10.7, allowing it to spend less time in configurations which require large torques, than the solution found using bootstrapping. The decreased goal costs come from the fact that the robot is already standing after only 6.0s (see Fig. 10.8e) rather than 8.0s as is the case with the less aggressive policy found by the bootstrapped optimization (see Fig. 10.8d). Around the solution space of feasible sit-to-stand policie parameters, the tracking cost has little impact on the total cost, but becomes more prominent when the policy fails.

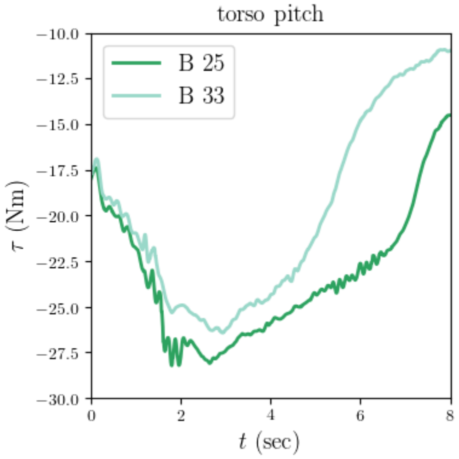


Figure 10.9 – Evolution of the torso pitch joint torques for the rollouts 25 and 33 in the bootstrapped case.

10.3 Conclusions

The test cases presented here show that TCFM is capable of efficiently producing successful whole-body behaviors on real systems by maximizing the compatibility and feasibility of the tasks. This is shown for a two common whole-body scenarios, reaching while balancing and standing up from sitting, and tested on two different whole-body control architectures. Through simulated and real-world experiments TCFM is clearly capable of resolving task incompatibilities and infeasibilities by accounting for and correcting un-modeled perturbations and modeling errors. TOTs are shown to be a robust alternative to GPTs as policy parameterizations, and require less prior task-related knowledge to implement. Using the compact parameterization afforded by TOTs, BO is shown to be more sample efficient than CMA-ES as an update strategy. This comes as no surprise for the relatively low-dimensional problems explored here, but remains to be tested for larger more complicated cases. While this formulation of TCFM with TOTs and BO, is an important first step towards bridging the gap between high-level planning and low-level control, there is still much work to be done to refine these techniques.

Part V

CONCLUSION

Chapter 11

Conclusions and Perspectives

Robots often perform unexpectedly when executing multiple simultaneous tasks, and the result is generally undesirable. Being able to automatically improve task execution based on trial-and-error is a key missing component in modern control architectures. Through a series of experiments, we have shown that TCFM is capable of maximizing task compatibility and feasibility on real humanoids, and thus fulfilling the role of this component. Nevertheless, a number of hurdles and limitations remain before this solution can be said to be truly autonomous and efficient. In this chapter, we discuss these limitations, some perspective solutions, and future work for improving on the methods developed here. We close this dissertation with some concluding remarks about the main takeaway messages from this work.

11.1 Is There Still a Problem with Robots?

As stated in the introduction of this study, the basic problem with complex robots is that when they perform multiple simultaneous tasks the resulting overall motion is typically not what was expected. This happens for many reasons, but it is primarily because the tasks are incompatible with one another and/or infeasible given the problem constraints. These incompatible and infeasible tasks are an artifact of nature of the control architecture used for modern whole-body control, and ultimately what is missing is the ability to automatically modify the tasks to account for the task servoing and whole-body controller layers (along with all of their parameters), and the modeling errors only observed at runtime on the real robot. We denoted this missing component Task Compatibility and Feasibility Maximization (TCFM), and it is depicted again here in Fig. 11.1.

Understanding Task Compatibility and Feasibility The goal of this dissertation was to construct the TCFM loop. To do so, the first step was to understand what it really means for tasks to be compatible and feasible. Therefore, using concepts of multi-objective optimization, a series of metrics were developed to measure the compatibility and feasibility of objective functions in Chapter 3. In Chapter 4, these metrics were applied to the whole-body control problem and notions of the evolution of task compatibility and feasibility were evoked. In Chapter 6, a series of known task incompatibilities and infeasibilities were used to test the metrics one by one, and it was found that only a few are actually useful for robotics.

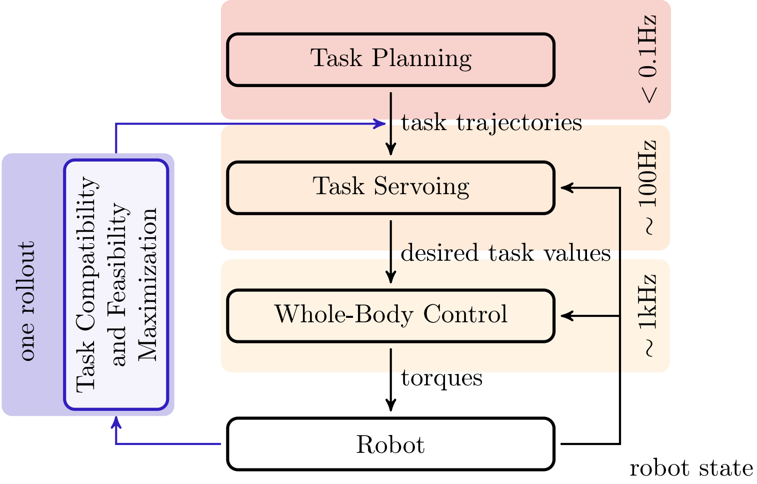


Figure 11.1 – The task compatibility and feasibility maximization loop proposed in this dissertation is designed to correct incompatible and infeasible tasks produced by this modern control architectures. This loop is one of trial-and-error learning, thus, the control frequency is replaced by the number of rollouts.

A Model-Based Solution Using the basic ideas gleaned from these metrics, we developed a whole-body Non-linear Model Predictive Control (NMPC) problem which could theoretically optimize tasks to render them both compatible and feasible while respecting their initial goals. It was quickly apparent, however, that this problem is likely intractable, even if only solved over a short preview horizon. Beyond the computational complexity, the issue of correcting for modeling errors goes unresolved with this technique.

A Model-Free Solution Given the difficulties posed by the model-based approach to task compatibility and feasibility maximization, in Chapter 5 Model-Free Policy Search (MFPS) was adopted. Here, the task servoing and whole-body control layers are treated as generic control policies, which are parameterized by the task trajectory parameters (e.g. waypoints). These parameters are then used to modify the policy in order to minimize a performance cost. The performance cost captures the side-effects (performance degradation) of task incompatibilities and infeasibilities. The result is the TCFM policy-search algorithm which combines model-free learning techniques with model-based control. With the same scenarios used to test the model-based metrics, in Chapter 7, it was shown that the model-free cost and TCFM are successful in both measuring and correcting the performance degradation due to incompatibility and infeasibility.

Validating on a Humanoid Having proven the concept of TCFM, the algorithm was tested on a simulation of the humanoid robot, iCub, performing multiple reaching tasks in Chapter 8. The results showed that the TCFM algorithm can work on a complex humanoid problem. Unfortunately, the high-dimensional Dynamic Movement Primitive (DMP) parameterization in association with the sample inefficient stochastic optimization update strategy, Policy Improvement though Black-Box optimization (PI^{BB}), used by the algorithm, require too many rollouts to be practical for real robots.

Improving the Algorithm In Chapter 9, the custom designed Gaussian Process Trajectory (GPT) parameterization was used with the Bayesian Optimization (BO) update strategy. The GPTs were shown to improve the robustness of tasks to temporary incompatibilities and infeasibilities, and their combination with BO showed marked improvement in the sample efficiency of TCFM, even for dynamically complex motions. However, GPTs required the waypoint times to be specified a priori and showed some undesirable numerical instabilities.

Proving it Works on a Real Robot To resolve these issues, in Chapter 10, Time-Optimal Trajectories (TOTs) were used as policy parameterizations, which only require waypoints and maximum velocities and accelerations. Using both CMA-ES and BO, it was shown that the new parameterization maintained sample efficiency and improved numerical robustness. The CMA-ES and BO update strategies were compared, and BO was shown to be nearly twice as sample efficient as CMA-ES, but less consistent in the number of rollouts needed for convergence. The TCFM algorithm was then tested using a different underlying control architecture, proving that TCFM is generic to the whole-body controller being used. Finally, it was shown that TCFM is robust and efficient enough to perform rollouts on a real robot and compensate for un-modeled real-world effects.

Problem Solved? So the question is, do robots still have a problem? Validating TCFM in both simulation and reality shows that the method is capable of "automatically" improving task trajectories to be more compatible and feasible, resulting behaviors being performed as expected. Therefore, while robots might start with incompatible and infeasible tasks, we now have ways to analyze and efficiently fix this problem. However, a number of limitations must be addressed before we can safely say that task compatibility and feasibility is no longer a problem for robots.

11.2 Limitations and Perspectives

In this section we discuss the limitations of the methods developed in this dissertation, and provide a few perspectives on potential solutions.

11.2.1 Evaluating the Metrics on a Humanoid

The model-based task compatibility and feasibility metrics were developed a posteriori in order to provide an alternative analytical quantification to the model-free TCFM costs. Unfortunately, there was insufficient time to explore the properties of these metrics on a humanoid problem. While the metrics were thoroughly tested on a 6 DoF manipulator in Chapter 6, it remains to be seen how well they can measure task compatibility and feasibility on a high-DoF system, like a humanoid. Most likely, the task compatibility measures which worked well for the manipulator would also work well for a higher dimensional system, but the calculation of the feasibility ellipsoid may be somewhat more complex given the number of inequality constraints. Furthermore, it is unclear how these metrics will behave with changing contacts. It is probable that the feasibility ellipsoid would change discretely given the addition or removal of contact constraints. Understanding and analyzing these metrics on a humanoid problem is important step in further understanding of the fundamental problem of task compatibility and feasibility.

11.2.2 Discrepancies Between κ^{NR} and κ_A^{NR}

In Chapter 6, we saw that the nuclear norm ratio metric, κ^{NR}, for task compatibility closely resembles what is intuitively expected of the compatibility between the tasks given the experimental setup. For instance, in Sec. 6.3.2, three tasks, which should have been perfectly compatible, were indicated as such by the κ^{NR} metric. However, the augmented formulation, κ_A^{NR}, which includes the equations of motion, indicated that the tasks are much less compatible than one would expect. This same result occurred in Sections 6.7.1, 7.2.2, and 7.4.3, where tasks, which should have been compatible, were indicated as mostly incompatible by κ_A^{NR}.

This is interesting because it raises two questions. The first question we must ask is, does it make sense to measure κ^{NR} at all? If an equality constraint, such as the equations of motion, is used in the whole-body controller then the tasks are forced to respect it. So, what does it mean to examine task compatibility without this constraint? Even more frustrating is the fact that the κ^{NR} seems to closely match what is expected in terms of task compatibility. Therefore, despite being theoretically useless, the κ^{NR} metric may be the most practical and intuitive task compatibility metric.

The second question is, why does κ_A^{NR} tell us that the tasks are incompatible with the dynamics, when they seem to be executed perfectly? In Chapter 6, we used this phenomenon to justify the whole-body controller "settling", where the task objective functions achieve equilibrium in the multi-objective optimization. If the tasks are somewhat incompatible with the equality constraints, then the equality constrained, prioritized optimum, $\boldsymbol{\chi}^*$, is different from the optimum found without the equality constraint. This means that the task objective functions are not perfectly optimized and will have a non-zero error, which is amplified by the task servoing layer creating new desired accelerations for the tasks. This continues until the objective functions all counter balance each other given their task servoing gains and their priorities. The result is the "settling" effect.

While the κ_A^{NR} values can help explain the settling of the whole-body controller, it is still unclear why seemingly compatible tasks are incompatible with the dynamics. Furthermore, the only observable pattern from the κ_A^{NR} metric is that compatibility, according to κ_A^{NR}, rises while the robot is moving. One possibility to solve this discrepancy would be to project the objectives into the equality constraints (the so-called reduced formalism [Salini, 2012]) and then compute only κ^{NR}. However, this technique may not work if a floating base and contacts are involved. Our conclusion then is that κ_A^{NR} merits deeper testing and analysis, because it seems to reveal aspects of task compatibility which have yet to be properly understood.

11.2.3 Postural Tasks are Probably a Bad Idea

Postural tasks are to redundant robotics, what the appendix is to the human body: a vestigial component which over time can cause problems. This may be a bit of an overstatement, but postural tasks are so common in robotics because they solve a lot of initial problems. They keep joints from moving randomly when they are redundant, and they keep the robot away from its joint limits. However, since they do not evolve with the motion, they tend to accumulate error and even though they are typically low priority tasks, they can impinge on other tasks given sufficient error and task servoing gains. Furthermore, from the compatibility analysis it is clear that the only task impeding the task set from attaining perfect compatibility (according to the κ^{NR} metric) is the postural task — and it is not even important. One solution for this might be to use a different error function for servoing the postural tasks like the ϵ-insensitive error function,

$$f_\epsilon(q) = \begin{cases} 0 & \text{if } |q| \leq \epsilon \\ |q| - \epsilon & \text{else} \end{cases}. \tag{11.1}$$

Thus applying zero error while the joints are in the middle of their range, and linearly increasing error as they approach their limits. Of course, this adds one more variable, ϵ, which must be tuned, and tuning is never ideal.

Another alternative, would be to learn the optimal posture for a given set of operational-space tasks. Using the operational-space task references as inputs and their resulting postures as outputs, a model could be learned to modify the postural references to be maximally compatible with the operational-space references.

11.2.4 Automating Task Parameterization

One of the key hinderances towards full automation of the TCFM algorithm is the selection of the task parameters which can be used to modify the policies. In all of the experiments, the tasks which are modified are selected by hand and their parameterization has to be custom designed to allow for "adjustable" parameters. In Chapter 8, DMPs are used to "automatically" parameterize the tasks but the number of kernel functions needs to be hand tuned in practical applications. This need to hand-select the policy parameters for TCFM is a key limitation and one which needs to be resolved.

One promising route to handle this is to use non-parametric function approximators to find the minimum number of parameters to accurately reconstruct initial task trajectories and do this for all tasks. TCFM could then be carried out on all tasks simultaneously, or at least those with a priority above a certain threshold. However, this would increase the dimension of the problem and hurt the sample efficiency of learning.

11.2.5 More Dimensions & Longer Problems

One of the main advantages of the TCFM approach is that fewer policy parameters are needed and more sample efficient update strategies can be used. In some ways, however, this is also a disadvantage. While TCFM is able to produce dynamically compatible and feasible whole-body motions with very few variables, it is unclear how far this can be pushed to longer and more complex motions. Most importantly, walking requires the robot to move around and the time and distance scales of walking may pose a challenge to maintaining the efficiency of TCFM. Simply adding more task trajectory waypoints to produce longer motions is the obvious solution, but the increase in dimensionality could render BO sample and data inefficient. The alternative then is to use stochastic optimization, but these strategies are sample inefficient.

One potential solution is to use step-based learning, rather than episodic learning. In step-based learning, the policies are updated at each time step rather than after the whole rollout [Deisenroth et al., 2013]. Since the full policy horizon is not used, step-based learning can sometimes produce myopic policies, but the length of these policies has less of an impact on the computational complexity. Using step-based learning could allow TCFM to be applied to long horizon problems, like walking, but it is not evident how this could be implemented.

11.2.6 Better Update Strategies

While BO is a useful and sample efficient update strategy, it is not without its problems. First and foremost is the fact that it is not efficient for problems with more than 10 dimensions. Every new sample increases the computational complexity, firstly by a factor of $O(n^3)$ for the inversion of K in (A.14), then linearly, $O(n)$, with the number of samples for each evaluation of the acquisition function. All this means is that optimization with BO becomes harder and harder as the size of the problem and number of samples are increased.

In addition to scaling issues, BO tends to aggressively explore the parameter space in the early rollouts and this can lead to slow convergence if there are local minima near the bounds. Depending on the chosen kernel function for the Gaussian Process (GP) used in BO, the initial search generally tests the parameter extrema because they have the highest variance. For the standing cases in Chapter 10, the difference between optimal and failing policy parameters is very small. When the failing parameters are tested in the initial rollouts, the GP, being undersupplied with data, associates a high cost to the parameter space around this failure. This causes the BO to explore the rest of the parameter space, until it has enough data to indicate a minimum near the initial failure. The result of this is a decrease in sample efficiency.

This brings us to the third weak point of BO, the meta-parameter selection process. BO is only as good as its surrogate GP function and if the kernel functions and their parameters are chosen poorly, then the latent cost function approximation may be inadequate. From the experiences in this work, the biggest issue is a lack of resolution around local optima. It is our experience that attaining consistent convergence with BO requires a certain degree of smoothness in the GP kernel meta-parameters and this leads to a smoothing of local optima, which hinders the ability of BO to locally improve solutions. This means that TCFM could be missing out on potential optima.

Improving on these points is a research domain in itself. However, a few points which are specific to TCFM could be immediately addressed. Firstly, starting BO with a single parameter-cost datum can lead to poor convergence properties. If the rollouts are being performed in simulation, then a good idea would be to randomly sample solutions around the initial policy parameters (à la CMA-ES) and, in parallel, roll the parameters out and evaluate their costs. This would bootstrap the surrogate function with more data and avoid the aforementioned convergence issues. Parallelizing simulations would also accelerate stochastic optimization techniques, such as CMA-ES, and this could be advantageous if the parameter dimension is too large for BO. Finally, when rolling out on the real robot, it is not easy to specify "safe" parameter bounds for TCFM. If rollouts are first performed in simulation and then on the real robot, as in Chapter 10, a simpler (and possibly wiser) idea is to search locally around the initial solution found by TCFM in simulation. This could be done with CMA-ES. The variance of the parameters, given by the BO surrogate function, could be used as the diagonal of the initial CMA-ES covariance matrix. The real rollouts would then be an unbounded local search around these parameters for incremental improvements. This would alleviate the fine tuning needed to find "safe" bounds.

11.2.7 Is Episodic Learning Closing the Loop?

At the outset of this study, it was stated that this TCFM layer should close the loop between motion execution and task trajectory planning. While this is true, it is a bit of a misnomer given that TCFM is based on episodic, or trial-and-error, learning. This means that the whole motion must be rolled out before any improvement can be seen, and, depending on the update strategy, there are no guarantees that the next rollout will be better. So, although we have linked the whole-body motion execution with the optimization of the task trajectories, we have not closed the loop in the classical control sense.

Additionally, there are two modes of operation for TCFM: rolling out in simulation and on the real robot. As shown throughout the experiments, performing TCFM in simulation can produce motions which are more compatible and feasible. Simulation-based TCFM can then be considered a form of open-loop planning, or filtering for the task planning layer, and the time it takes to optimize the tasks can be computed as, rollout time × no. rollouts. For example, the standing test case from Sec. 10.2.4 we have, $\sim 10\frac{\text{sec}}{\text{rollout}} \times 25\text{rollouts} \simeq 250\text{sec}$. By improving on the speed of the rollout simulations, the time needed to optimize the tasks can be reduced, and ideally this time would be as small as possible.

Performing TCFM on the real robot can account for phenomena which were not simulated and correct for modeling errors. In this case, the loop between task execution and task definition is closed in the sense that real output data is used as feedback to modify the inputs. However, because an entire rollout is needed, there is no notion of control loop frequency. It is therefore more pertinent to discuss the number of rollouts needed to optimize the tasks, rather than the control frequency of the optimization, because this is a learning loop and not a control loop. Since learning requires some amount of exploration, i.e. testing suboptimal control laws, and control loops are designed to tend asymptotically towards stable optima, we are inclined to say that episodic learning is not equivalent to a control loop. Nevertheless, episodic learning provides a powerful alternative to scenarios where developing an optimal control law is prohibitive.

11.3 Future Work

Having looked at some of the various questions and limitations of this work, we now explore where to go next.

11.3.1 Implement the NMPC Problem

Despite its complexity, the NMPC problem developed to maximize task compatibility and feasibility could eventually be tractable given the advances in NMPC techniques in recent years [Koenemann et al., 2015]. Implementation details of this problem remain to be fleshed out, but an analytical alternative to TCFM, or in addition to TCFM, would be a powerful tool in solving task incompatibilities and infeasibilities. In theory, it could be used in the place of doing TCFM in simulation as the filter for the task trajectory planning. Of course, the issues associated with modeling errors would still be a problem, so developing the NMPC problem is not a silver bullet for maximizing task compatibility and feasibility. Thus, keeping the layer of TCFM for trial-and-error learning on the real robot could provide a complete solution.

11.3.2 TCFM Using Model-Based Metrics as the Cost Functions

Rather than use the model-free cost functions developed in Chapter 5, an interesting avenue would be to use the model-based metrics as cost functions for the TCFM. There would be two primary advantages to this approach. The first is that the MFPS would be optimizing the task compatibility and feasibility directly, rather than indirectly through measuring its side-effects. This could improve on the speed of convergence and the quality of the resulting solutions. The second advantage is that the metrics could be used as an automated way of determining when a task needs to be optimized. If the compatibility of a task with the other tasks, or the feasibility of a task with the constraints set, exceeds a certain threshold, then this task could be parameterized and its metric minimized using MFPS. However, it is not entirely clear which thresholds would be best suited to this. The downsides to using the model-based metrics are that a greater understanding of the underlying whole-body control architecture is needed, and that the metrics are not as trivial to calculate as the model-free costs.

11.3.3 Reusing Surrogate Models Across Scenarios

The main factor in the sample efficiency of BO is that the search is guided by the surrogate function which is learned as new samples are obtained. The more samples, the better the surrogate function describes the latent cost function. The question then, is what to do with the surrogate when the TCFM is finished? In this work, the surrogate was used to bootstrap the TCFM rollouts on the real robot, but more can be done. Specifically, it would be interesting to investigate how to reuse these surrogate functions across task sets and learn a global model of task compatibility and feasibility. Doing this efficiently however, is not immediately obvious. Should the parameters from each task be included in the GP model? If so, then the dimension of the GP may make it unusable. If surrogate functions are learned for each specific task scenario, then they must be combined in some meaningful way. One option could be to learn a repertoire of surrogate functions and reuse them based on contextual information. The problem then becomes how to parameterize the context.

11.3.4 A Deep Model of Task Compatibility and Feasibility

Although we started this study essentially saying end-to-end learning would not work on real systems, we do recognize the power of deep models. In this regard, one exciting perspective for task compatibility and feasibility would be to record the states of key parts of the robot, i.e. the CoM, hands, feet, and robot state, and the corresponding model-based task compatibility and feasibility metrics at every time step. Using this data a deep model could be learned over a horizon of time steps with the states as inputs and the metrics as outputs. Assuming the model can accurately learn this relationship, then is could be used to optimize the tasks which manage these states to improve task compatibility and feasibility. We assume that deep models would be necessary because of the size of the input space and the complexity of the underlying mapping from task trajectories to execution. This model would improve over time and could be used when the predictions are sufficiently close to the measured metrics.

11.4 Closing Remarks

We began this work with the simple objective of trying to understand why robots do not do what is asked of them. By digging into this question, we realized that when robots do not perform as expected, it is simply because what is being asked of them is not possible. Thus, we set out to develop the tools techniques, and understanding, necessary to handle these situations. From this, three main contributions were presented:

1. We formally posed the problem of task compatibility and feasibility.
2. We determined model-based and model-free ways of measuring these phenomena.
3. We developed a MFPS algorithm which exploits model-based whole-body control to efficiently maximize task compatibility and feasibility.

Understanding the issue of task compatibility and feasibility is a key hurdle in the race to more robust and capable control architectures. There was a time when robots were mechanically unequipped for the problems they were made for, but this is no longer the case. Modern humanoids are capable of handling much more than modern control can throw at them and this is a shortcoming which must be compensated. Part of this problem is the issue of task compatibility and feasibility. We are limited in how well we can plan multiple tasks and these limitations are propagated through the control architecture. Try as we might to compensate for this poor planning, better task servoing, tuning gains, priorities, etc. the root of the problem lies in the definitions of the tasks themselves. But rather than lump this burden on to high-level planning, robots should have the tools to automatically convert poorly planned tasks into tasks which they can achieve. Whether this is done using model-based, model-free, or a combination of the two, only time will tell, but in this work we show that it can be done, and done in a way which is practical for real robots. We believe that keeping this practicality in mind is important because problems typically arise when moving from simulation to reality.

To conclude this dissertation, we leave the reader with a three take home messages.

I The need for prioritization in multi-task control is a direct indicator that something is wrong. Tasks which are compatible and feasible do not need priorities. Why should we expect the robot to execute its motion if it is mathematically impossible? We do not mean to say that prioritization is useless, only that it is a sign of our inability to properly plan tasks.

II Adding layers of learning to modern whole-body controllers has the potential to solve hard problems. By exploiting whole-body control architectures, rather than ignoring them, learning can be abstracted to a few task related variables. These small dimensional spaces mean that real robots can learn useful control policies without the need for thousands of rollouts. If we are to learn on real platforms, then this is critical.

III Finally, the main takeaway from this work is that understanding and resolving task incompatibility and infeasibility is the next step towards higher-levels of learning and reasoning. Closing loops in control is the only way to safely move outwards towards more complex control objectives, and as of now the loop between motion execution and task planning is open. We have made progress towards closing it, but much work remains to be done before we are confident that when we ask a robot to do something, it really does it.

Part VI

BACK MATTER

Appendix A

Appendix

A.1 Task and Constraint Formula References

task type	mathematical formulae	eq. reference
operational-space acceleration	$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} - \ddot{\boldsymbol{\xi}}_i^{\text{des}} \right\|_2^2 = \left\| E^{\ddot{\boldsymbol{\xi}}}\boldsymbol{\chi} - \mathbf{f}^{\ddot{\boldsymbol{\xi}}} \right\|_2^2$ $E^{\ddot{\boldsymbol{\xi}}} = \begin{bmatrix} J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix}$ $\mathbf{f}^{\ddot{\boldsymbol{\xi}}} = \ddot{\boldsymbol{\xi}}_i^{\text{des}} - \dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}$	(2.11) (2.15) (2.13) (2.14)
joint-space acceleration	$f_i^{\dot{\boldsymbol{\nu}}} = \left\| \dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_i^{\text{des}} \right\|_2^2 = \left\| E^{\dot{\boldsymbol{\nu}}}\boldsymbol{\chi} - \mathbf{f}^{\dot{\boldsymbol{\nu}}} \right\|_2^2$ $E^{\dot{\boldsymbol{\nu}}} = \begin{bmatrix} I & \mathbf{0} \end{bmatrix}$ $\mathbf{f}^{\dot{\boldsymbol{\nu}}} = \dot{\boldsymbol{\nu}}_i^{\text{des}}$	(2.16) (2.19) (2.17) (2.18)
wrench	$f_i^{\boldsymbol{\omega}} = \left\| {}^e\boldsymbol{\omega}_i - {}^e\boldsymbol{\omega}_i^{\text{des}} \right\|_2^2 = \left\| E^{\boldsymbol{\omega}}\boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\omega}} \right\|_2^2$ $E^{\boldsymbol{\omega}} = \begin{bmatrix} \mathbf{0} & S_i^{\boldsymbol{\omega}} \end{bmatrix}$ $\mathbf{f}^{\boldsymbol{\omega}} = {}^e\boldsymbol{\omega}_i^{\text{des}}$	(2.21) (2.26) (2.23) (2.24)
torque	$f^{\boldsymbol{\tau}} = \left\| \boldsymbol{\tau} - \boldsymbol{\tau}^{\text{des}} \right\|_2^2 = \left\| E^{\boldsymbol{\tau}}\boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\tau}} \right\|_2^2$ $E^{\boldsymbol{\tau}} = \begin{bmatrix} \mathbf{0} & S^{\top} & \mathbf{0} \end{bmatrix}$ $\mathbf{f}^{\boldsymbol{\tau}} = \boldsymbol{\tau}^{\text{des}}$	(2.28) (2.32) (2.30) (2.31)

Table A.1 – A summary of the different task formulations for the whole-body controller. The original equation references for each task are indicated in the “reference” column.

constraint type	mathematical formulae	eq. reference
equations of motion	$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \underbrace{C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{q})}_{\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})} = S^\top \boldsymbol{\tau} + {}^eJ^\top(\boldsymbol{q})^e\boldsymbol{\omega}$ $\underbrace{\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^eJ^\top(\boldsymbol{q}) \end{bmatrix}}_{A^d} \boldsymbol{\chi} = \underbrace{\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})}_{\boldsymbol{b}^d}$	(2.6) (2.33)
torque limits	$\boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\max}$ $\underbrace{\begin{bmatrix} \mathbf{0} & S^\top & \mathbf{0} \\ \mathbf{0} & -S^\top & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{\tau}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \boldsymbol{\tau}_{\max} \\ -\boldsymbol{\tau}_{\min} \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{\tau}}}$	(2.34) (2.35)
joint position limits	$\boldsymbol{q}_{\min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{\max}$ $\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{q}}} \boldsymbol{\chi} \leq \underbrace{\frac{2}{h^2} \begin{bmatrix} \boldsymbol{q}_{\max} - (\boldsymbol{q} + h\boldsymbol{\nu}) \\ -\left[\boldsymbol{q}_{\min} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{q}}}$	(2.36) (2.41)
joint velocity limits	$\boldsymbol{\nu}_{\min} \leq \boldsymbol{\nu} \leq \boldsymbol{\nu}_{\max}$ $\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{\nu}}} \boldsymbol{\chi} \leq \underbrace{\frac{1}{h} \begin{bmatrix} \boldsymbol{\nu}_{\max} - \boldsymbol{\nu} \\ -\left(\boldsymbol{\nu}_{\min} - \boldsymbol{\nu}\right) \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{\nu}}}$	(2.37) (2.42)
joint acceleration limits	$\dot{\boldsymbol{\nu}}_{\min} \leq \dot{\boldsymbol{\nu}} \leq \dot{\boldsymbol{\nu}}_{\max}$ $\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\dot{\boldsymbol{\nu}}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \dot{\boldsymbol{\nu}}_{\max} \\ -\dot{\boldsymbol{\nu}}_{\min} \end{bmatrix}}_{\boldsymbol{h}^{\dot{\boldsymbol{\nu}}}}$	(2.38) (2.43)
friction contacts	${}^FJ_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + {}^F\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} = \mathbf{0}$ ${}^FC_i{}^F\boldsymbol{\omega}_i \leq \mathbf{0}$ $\underbrace{\begin{bmatrix} {}^FJ_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix}}_{A^{\boldsymbol{\omega}}} \boldsymbol{\chi} = \underbrace{-{}^F\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}}_{\boldsymbol{b}^{\boldsymbol{\omega}}}$ $\underbrace{\begin{bmatrix} \mathbf{0} & {}^FC_iS_i^F \end{bmatrix}}_{G^{\boldsymbol{\omega}}} \boldsymbol{\chi} \leq \underbrace{\mathbf{0}}_{\boldsymbol{h}^{\boldsymbol{\omega}}}$	(2.44) (2.45) (2.47) (2.48)

Table A.2 – A summary of the different constraint formulations for the whole-body controller. The original equation references for each constraint are indicated in the “reference” column.

A.2 Objective Compatibility and Feasibility Metric References

symbol	mathematical formula	description
κ^{NR}	$\frac{\left\Vert \tilde{E} \right\Vert_*}{\left\Vert \left[\tilde{E} \vert \tilde{\mathbf{f}}\right] \right\Vert_*}$	Nuclear norm Ratio (NR): The ratio of the nuclear norms of the linear algebraic system created by the objective functions.
κ^{OD}	$\sum_{i=1}^{n_o} \kappa_i^{\mathrm{OD}} = \sum_{i=1}^{n_o} \Vert \boldsymbol{x}^* - \boldsymbol{x}_i^* \Vert_2$	Optimum Distance (OD): The sum of the distances between each of the individual objective optima and $\boldsymbol{x}^*$.
κ^{OC}	$\sum_{i=1}^{n_o} \kappa_i^{\mathrm{OC}} = \sum_{i=1}^{n_o} w_i f_i(\boldsymbol{x}^*)$	Optimum Cost (OC): The sum of the costs of each objective function evaluated at $\boldsymbol{x}^*$.
κ^{EV}	$\det(B_\kappa) = \prod_{i=1}^{n} \sigma_i$	Ellipsoid Volume (EV): The volume of the compatibility ellipsoid, as calculated by the determinant of its B matrix, or equivalently, the product of its eigenvalue scale factors, $\sigma_i \in i \dots n$.
κ^{ED}	$\sum_{i=1}^{n_o} \kappa_i^{\mathrm{ED}} = \sum_{i=1}^{n_o} \Vert \boldsymbol{c}_\kappa - \boldsymbol{x}_i^* \Vert_2$	Ellipsoid Distance (ED): The sum of the distances of the individual objective optima to the center of the compatibility ellipsoid.

Table A.3 – Objective compatibility metric summary. The variable n_o is the number of objective functions, n the dimension of the optimization variable, $\boldsymbol{x}$, $\boldsymbol{c}_\kappa$ the compatibility ellipsoid center, B_κ, the compatibility ellipsoid matrix, and $\boldsymbol{x}^*$ is the overall global unconstrained problem optimum.

symbol	mathematical formula	description
ϕ^{OD}	$\sum_{i=1}^{n_o} \phi_i^{\mathrm{OD}} = \sum_{i=1}^{n_o} \Vert \boldsymbol{c}_\phi - \boldsymbol{x}_i^* \Vert_2$	Optima Distance (OD): The sum of the distances between each of feasibility ellipsoid center $\boldsymbol{c}_\phi$.
ϕ^{ED}	$\Vert \boldsymbol{c}_\kappa - \boldsymbol{c}_\phi \Vert_2$	Ellipsoid Distance (ED): The distance between the compatibility and feasibility ellipsoid centers.
ϕ^{EE}	$\sum_{i=1}^{n_o} \phi_i^{\mathrm{EE}} = \sum_{i=1}^{n_o} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{x}_i^* - \boldsymbol{c}_\phi\right)$	Ellipsoid Evaluation (EE): Sum of the results of the feasibility ellipsoid inequality equation, (3.63), at each $\boldsymbol{x}_i^*$.
ϕ^{EE^*}	$\left(\boldsymbol{x}^* - \boldsymbol{c}_\phi\right)^\top P_\phi^{-1} \left(\boldsymbol{x}^* - \boldsymbol{c}_\phi\right)$	Ellipsoid Evaluation (EE*): Result of the feasibility ellipsoid inequality equation, (3.63), at each $\boldsymbol{x}^*$.
ϕ^{EV}	$\det(B_\phi) = \prod_{i=1}^{n_c} \sigma_i$	Ellipsoid Volume (EV): The volume of the feasibility ellipsoid, as calculated by the determinant of its B matrix, or equivalently, the product of its eigenvalue scale factors, $\sigma_i \in i \dots n_c$.

Table A.4 – Objective feasibility metric summary. The variable n_o is the number of objective functions, n_c is the number of constraint equations, n the dimension of the optimization variable, $\boldsymbol{x}$, $\boldsymbol{c}_\kappa$ and $\boldsymbol{c}_\phi$, the compatibility and feasibility ellipsoid centers, P_ϕ and B_ϕ, the feasibility ellipsoid matrices, and $\boldsymbol{x}^*$ is the overall global unconstrained problem optimum.

A.3 Model-Free Task Compatibility and Feasibility Cost Function References

symbol	mathematical formula	description
j^{T}	$\sum_{k=0}^{N} \lVert \boldsymbol{\epsilon}(k) \rVert_2^2$	Tracking Cost (T): Measures the tracking error, in a Euclidean sense, of the position of a task over the entire policy rollout.
j^{G}	$\sum_{k=0}^{N} \frac{kh}{t_{n_\lambda}} \lVert \boldsymbol{\epsilon}^{\mathrm{goal}}(k) \rVert_2^2$	Goal Cost (G): Measures a task's distance to its goal position, in a Euclidean sense, over the entire policy rollout.
j^{E}	$\sum_{k=0}^{N} \lVert \boldsymbol{\tau}(k) \rVert_2^2$	Energy Cost (E): Measures the magnitude of the control input necessary to execute the policy.
j^{TP}	$\sum_{i=1}^{n_{\mathrm{task}}} w_i j_i^{\mathrm{TP}} = \sum_{i=1}^{n_{\mathrm{task}}} w_i \left(j_i^{\mathrm{T}} + j_i^{\mathrm{G}} \right)$	Task Performance Cost (TP): The combination of the tracking and goal costs of each task, accumulated in a summation weighted by their priorities.
j^{P}	$\frac{j^{\mathrm{TP}} + \beta j^{\mathrm{E}}}{t_{\boldsymbol{\pi}}^{\mathrm{end}}}$	Performance Cost (P): Overall measure of the performance of a policy, which encompasses all effects of task incompatibilities and infeasibilities.

Table A.5 – A summary of the various cost functions used to evaluate policies. These functions are designed to capture various aspects of whole-body control performance degradation resulting from task incompatibilities and infeasibilities. The term k indicates the control time step, and N the total number of time steps. The actual total duration of the rollout is defined as $t_{\boldsymbol{\pi}}^{\mathrm{end}} = Nh$, where h is the control sampling period. The term $\boldsymbol{\epsilon}(k)$ is the task position tracking error at time k, and $\boldsymbol{\epsilon}^{\mathrm{goal}}(k)$, is the difference between the current task position and its last waypoint $\boldsymbol{\lambda}_{n_\lambda}$ at the k^{th} time step.

A.4 Dynamic Movement Primitives

Dynamic Movement Primitives, or DMPs, are constructed through the combination of one or more dynamic systems and a non-linear forcing term which allows their passage through space and time to be arbitrarily regulated. Following [Kulvicius et al., 2012] and using the same scheme described by [Stulp, 2014], the following DMP formulation is used in this work,

$$\begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{z}} \\ \dot{\boldsymbol{y}}^{\gamma} \\ \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}/\tau \\ ((K(\boldsymbol{y}^{\gamma} - \boldsymbol{y}) - D\boldsymbol{z}) + v \cdot \boldsymbol{f}(x, \boldsymbol{\theta}))/\tau \\ -\alpha_{\gamma}(\boldsymbol{\gamma} - \boldsymbol{y}^{\gamma}) \\ 1/\tau \\ -\alpha_v v(1 - v/v_{max}) \end{bmatrix} \begin{matrix} (a) \\ (b) \\ (c) \\ (d) \\ (e) \end{matrix} . \tag{A.1}$$

The basic building block of this DMP is the spring-damper dynamic system, shown in lines (a) and (b), in which $[\, \boldsymbol{y}^T, \dot{\boldsymbol{y}}^T \,]^T$ is the state of the system, and $\ddot{\boldsymbol{y}} = \dot{\boldsymbol{z}}/\tau$ is the system output. The vector $\boldsymbol{y}$ typically contains the position and possibly orientation for some frame of reference. The terms K and D represent the stiffness and damping coefficients respectively, and are generally set for critical damping with $D = 2\sqrt{K}$. The attractor, or goal term, $\boldsymbol{\gamma}$, forces the system to converge to a specific point, and τ is proportional to the duration of the movement the DMP is approximating. Finally, the forcing term, $\boldsymbol{f}$, parameterizes the system's evolution from start to finish and is an open-loop controller which is a function of its parameters, $\boldsymbol{\theta}$, and an auxiliary dynamic system, x.

To make sure the open-loop forcing term degrades to zero near $\boldsymbol{\gamma}$, the sigmoid function gating system, shown in line (e), is used to regulate $\boldsymbol{f}$ where, α_v and v_{max}, are factors which regulate the decay rate and time of the system. The state x is governed by the phase system[1], in line (d), which mimics the passage of time. The exponential goal system, in line (c), is also added to prevent high initial accelerations, and causes the virtual goal, $\boldsymbol{y}^{\gamma}$, to decay exponentially towards the true goal point, $\boldsymbol{\gamma}$.

The output of this DMP is a set of rates of change for the system states, which provide a trajectory controller in the DMP's parameter space. The reader is directed to [Kulvicius et al., 2012; Ijspeert et al., 2013] and [Stulp, 2014] for a more in depth discussion of each of these systems.

The forcing term, $\boldsymbol{f}$, is formulated as a non-linearly weighted combination of linear basis functions which allows it to approximate non-linear functions using regression techniques (see [Sigaud et al., 2011] for details),

$$\boldsymbol{f}(x, \boldsymbol{\theta}) = \sum_i \psi_i(x)(a_i x + b_i). \tag{A.2}$$

Here the affine basis function $a_i x + b_i$ is weighted by a non-linear Gaussian,

$$\psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right). \tag{A.3}$$

In total, the forcing term is parametrized by four values: the affine slopes and offsets, $\boldsymbol{a}$ and $\boldsymbol{b}$, as well as the Gaussian centers and widths, $\boldsymbol{c}$ and $\boldsymbol{\sigma^2}$,

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \\ \boldsymbol{\sigma^2} \end{bmatrix}. \tag{A.4}$$

These parameters can be adjusted to approximate any demonstrated or simulated trajectory via Linearly Weighted Regression (LWR) developed by [Schaal and Atkeson, 1998], or some variant of this technique such as those shown by [Munzer et al., 2014].

---

[1] This system is referred to as the "canonical" system by [Ijspeert et al., 2013].

A.5 Gaussian Process

Gaussian Process, or GPs, are widely used in machine learning for modeling complex data sets [Rasmussen and Williams, 2006]. They belong to the family of kernel based supervised learning techniques and are used to stochastically model relationships between a vector of inputs, $\boldsymbol{a}$, and their outputs, $\boldsymbol{b}$, by defining a distribution over mean, $m(\hat{a}, \boldsymbol{a})$ and covariance, $k(\hat{a}, \boldsymbol{a})$, functions. In the following demonstration, the inputs and outputs of the GP are unidimensional.

$$\hat{b} = f(\hat{a}) \quad \text{where} \quad f \sim \mathcal{GP}(m(\hat{a}, \boldsymbol{a}), k(\hat{a}, \boldsymbol{a})) \ . \tag{A.5}$$

In (A.5), f is the function to be approximated and is distributed as a GP over the mean and covariance functions. The term $\hat{a}$ indicates an unobserved input, while $\boldsymbol{a}$ and $\boldsymbol{b}$ are the observed training data. Henceforth, the hat symbol, $\hat{\bullet}$, is used to indicate and unobserved, or unmeasured, variable. The choice of $m(\hat{a}, \boldsymbol{a})$ and $k(\hat{a}, \boldsymbol{a})$ depends on the problem at hand, and a number of options exist [Rasmussen and Williams, 2006]. In this work we use a zero mean function,

$$m(\hat{a}, \boldsymbol{a}) = 0 \ , \tag{A.6}$$

and the squared exponential covariance kernel function,

$$k(\hat{a}, a_i) = \sigma_k \exp\left[\frac{-(\hat{a} - a_i)^2}{2l_k^2}\right] . \tag{A.7}$$

Typically one kernel is centered on each input datum, but a subset may also be used. The variable σ_k is the maximum allowable covariance. The term l_k is the length parameter, which influences how much adjacent kernel centers, a_i, influence one another. These terms, known as meta-parameters, play an integral role in the resulting GP model. They can be hand-selected based on prior model knowledge, optimized over the training data, or calculated through heuristics, as is done here. To simplify we use the notation,

$$f \sim \mathcal{GP}(0, k(\hat{a}, \boldsymbol{a})) = \mathcal{GP}(\hat{a}, \boldsymbol{a}, \boldsymbol{b}) \ . \tag{A.8}$$

The GP represents our prior knowledge of f. By assuming that the posterior probability of $\hat{b}$, given $\hat{a}$ and the training data, is distributed as a zero mean Gaussian with noise, the joint distribution of the prior and posterior can be expressed as,

$$\begin{bmatrix} \boldsymbol{b} \\ \hat{b} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_* & K_{**} \end{bmatrix} \right) \ . \tag{A.9}$$

For $[a_1, a_2, \ldots a_n] \in \boldsymbol{a}$, the terms, $K(\boldsymbol{a})$, $K_*(\hat{a}, \boldsymbol{a})$ and $K_{**}(\hat{a})$, are calculated as follows,

$$K(\boldsymbol{a}) = \begin{bmatrix} k(a_1, a_1) & k(a_1, a_2) & \cdots & k(a_1, a_n) \\ k(a_2, a_1) & k(a_2, a_2) & \cdots & k(a_2, a_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(a_n, a_1) & k(a_n, a_2) & \cdots & k(a_n, a_n) \end{bmatrix} , \tag{A.10}$$

$$K_*(\hat{a}, \boldsymbol{a}) = \begin{bmatrix} k(\hat{a}, a_1) & k(\hat{a}, a_2) & \cdots & k(\hat{a}, a_n) \end{bmatrix} \tag{A.11}$$

and

$$K_{**}(\hat{a}) = k(\hat{a}, \hat{a}) \ . \tag{A.12}$$

The dependencies of these terms are omitted in (A.9) to condense the notation. From (A.9), based on [Rasmussen and Williams, 2006] (p.200), the conditional distribution can be written as,

$$P(\hat{b} \mid \hat{a}, \boldsymbol{a}, \boldsymbol{b}) \sim \mathcal{N}(\hat{b}_{\mu}, \hat{b}_{\sigma^2}) \ , \tag{A.13}$$

where,

$$\begin{bmatrix} \hat{b}_{\mu}(\hat{a}) \\ \hat{b}_{\sigma^2}(\hat{a}) \end{bmatrix} = \begin{bmatrix} K_* K^{-1} \boldsymbol{b}^T \\ K_{**} - K_* K^{-1} K_*^T \end{bmatrix} = \mathcal{GP}(\hat{a}, \boldsymbol{a}, \boldsymbol{b}) \ . \tag{A.14}$$

Variables $\hat{b}_{\mu}(\hat{a})$ and $\hat{b}_{\sigma^2}(\hat{a})$ are the estimated sample mean and variance of $\hat{b}$. Evaluating (A.14) is commonly referred to as GP regression.

A.6 Bayesian Optimization

BO is a non-linear and non-convex optimization technique, popular in fields where evaluating a cost function is burdensome, as in robotics. As with any optimization, the goal of BO is to minimize a cost function, f,

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \; f(\boldsymbol{\omega}) \;, \tag{A.15}$$

with optimization variables, $\boldsymbol{\omega}$. Unlike other optimization techniques, rather than randomly sample the cost function to estimate gradient information, BO constructs a global model of the cost function based on the observed samples $\left[\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots \boldsymbol{\omega}_n\right] \in \Omega$ and their corresponding costs, $\left[j_1, j_2, \ldots j_n\right] \in \boldsymbol{j}$. This model, commonly referred to as the surrogate function, is then used to determine the next set of parameters to test, which both minimize the estimated cost and improve the model accuracy.

The tool used to build this model is known as a Gaussian process (GP). For an unobserved optimization variable set, $\hat{\boldsymbol{\omega}}$, it tells us their predicted cost mean, $\hat{j}_\mu$ and variance, $\hat{j}_{\sigma^2}$[2].

$$\begin{bmatrix} \hat{j}_\mu \\ \hat{j}_{\sigma^2} \end{bmatrix} = \begin{bmatrix} K_* K^{-1} \boldsymbol{j}^\top \\ K_{**} - K_* K^{-1} K_*^\top \end{bmatrix} = \mathcal{GP}(\hat{\boldsymbol{\omega}}, \Omega, \boldsymbol{j}) \;. \tag{A.16}$$

In (A.16), the K matrices are determined by evaluating a mean and covariance function over the training and unobserved data. Here we use the zero mean and squared exponential covariance kernel functions [Rasmussen and Williams, 2006]. The term K is the projection of each training datum, $\boldsymbol{\omega}_i$, into the kernel space, K_* is the projection of $\hat{\boldsymbol{\omega}}$ into the kernel space, and K_{**} is the projection of $\hat{\boldsymbol{\omega}}$ into a kernel defined by itself. Evaluating (A.16) is commonly referred to as GP regression and implementation details can be found in [Rasmussen and Williams, 2006].

Given the surrogate function, the next step in BO is to determine the optimal set of parameters, $\hat{\boldsymbol{\omega}}^*$, to sample on the next evaluation of the cost function. To do so, the exploitation and exploration of the surrogate function must be balanced. Simply choosing the minimum predicted value may leave us in a local minimum, and areas of high variance should be explored to improve the surrogate's accuracy. An acquisition function is therefore used to balance the exploration and exploitation of the knowledge about the cost function [Brochu et al., 2010]. It is a function of the estimated cost means and variances, $f_{acq}(\hat{\boldsymbol{j}}_\mu, \hat{\boldsymbol{j}}_{\sigma^2})$. The terms $\hat{\boldsymbol{j}}_\mu$ and $\hat{\boldsymbol{j}}_{\sigma^2}$ are determined by evaluating the GP surrogate function over the entire input space, $\hat{\Omega}$,

$$\begin{bmatrix} \hat{\boldsymbol{j}}_\mu \\ \hat{\boldsymbol{j}}_{\sigma^2} \end{bmatrix} = \mathcal{GP}(\hat{\Omega}, \Omega, \boldsymbol{j}) \;. \tag{A.17}$$

To obtain $\hat{\Omega}$, the search space must be bounded and discretized. This means that the search space grows as m^n, where m is the number of increments between bounds, and n is the dimension of $\boldsymbol{\omega} \in \mathbb{R}^{n \times 1}$. This is the principal limitation of BO. Recently, a method for circumventing this limitation has been proposed in [Snoek et al., 2015], but is not used in this work.

Minimizing f_{acq} w.r.t. $\boldsymbol{\omega}$, we find the next best set of optimization variables to test, $\hat{\boldsymbol{\omega}}^*$, which balance the exploration and exploitation of f,

$$\hat{\boldsymbol{\omega}}^* = \arg\min_{\boldsymbol{\omega}} \; f_{acq}(\hat{\boldsymbol{j}}_\mu, \hat{\boldsymbol{j}}_{\sigma^2}) \;. \tag{A.18}$$

The real cost function of our problem is sampled by evaluating the optimal variable set, $\hat{\boldsymbol{\omega}}^*$. In robotics, this corresponds to executing the task(s) and calculating their cost. Given this new data, $\boldsymbol{\omega}^*$, and its cost, j^*, the surrogate is updated[3]. The update consists simply of concatenating the new data to the old data and reevaluating (A.16). The optimization concludes when some problem-dependent convergence criterion is met.

---

[2] Henceforth, the hat symbol, $\hat{\bullet}$, is used to indicate and unobserved, or unmeasured, variable.

[3] Note the omission of the hat symbol, $\hat{\bullet}$, indicating that $\boldsymbol{\omega}^*$ and j^* have been observed.

Index

absolute objective feasibility, 66
acceleration tasks, 45
acquisition function, 95
actions, 82
Adaptive control, 35
agent, 82
Augmented Nuclear norm Ratio, 74

Bayesian Optimization, 38, 95
bilateral contacts, 48

Center of Mass, 29
Center of Pressure, 144
compatibility ellipsoid, 65
compatible, 32
configuration space, 28
controllability, 33
convex optimization, 56
Covariance Matrix Adaptation - Evolutionary Strategy, 95

Deep Deterministic Policy Gradient, 38
Deep Reinforcement Learning, 38
Degrees of Freedom, 30
derivative-free methods, 94
direct collocation, 35
discrete time control, 82
dynamic manipulability, 34
Dynamic Movement Primitives, 37

Ellipsoid Distance, 66
Ellipsoid Distance (ED), 68
Ellipsoid Evaluation (EE), 69
Ellipsoid Volume, 65
Ellipsoid Volume (EV), 69
end-to-end, 38
Energy (E) Cost, 90
environment, 82
episodic learning, 37, 92

feasible, 32
force ellipsoid, 34

Gaussian Process, 86
Gaussian Process Trajectory, 86
Goal (G) Cost, 89
GPT, 86

hierarchical prioritization, 49
humanoids, 27
hybrid prioritization strategy, 50

incompatible, 60
inconsistent, 61
Inverse Dynamics, 30
Inverse Velocity Kinematics, 30
isotropic points, 33

Karush Kuhn Tucker (KKT) equations, 57
kinodynamic planning, 28

Löwner John Ellipsoid, 64
Lagrange multipliers, 57
lexicographic optimization, 58
Linear Quadratic Gaussian, 35
Linear Time-Invariant, 29
Locally Weighted Regression, 132
Lower Confidence Bounds, 96

manipulability measure, 34
Model Predictive Control, 29
Model-Free Policy Search, 36
motion planning, 28
multi-objective optimization, 58

neural networks, 38
Non-linear MPC, 35
nuclear norm, 63
Nuclear norm Ratio (NR), 63

objective, 56

objective compatibility, 60
objective feasibility, 60, 66
Optima Distance (OD), 68
optimization variable, 56
Optimum Cost, 63
Optimum Distance, 63

Performance (P) Cost, 91
policies, 36
policy, 82
Policy Improvement though Black-Box optimization, 94
Proportional-Integral-Derivative, 29

Quadratic Program, 31

random search, 94
Rapidly exploring Random Tree, 28
Robust control, 35
rollout, 36

sample efficiency, 94
Semidefinite Programming, 64
Sequential Linear Quadratic, 35
state, 82
stochastic policies, 87
surrogate function, 95
task, 45
task compatibility, 34
Task Compatibility and Feasibility Maximization, 39, 82, 92
Task Performance (TP) Cost, 91
task planning, 27
task servoing, 28
Time-Optimal Path Parameterization, 29
Time-Optimal Trajectory, 88
torque tasks, 46
TOT, 88
Total Task Performance Cost, 91
trace norm, 63
Tracking (T) Cost, 89
transition dynamics, 82

unilateral contacts, 48

variance modulated prioritization, 87
velocity ellipsoid, 34
viability, 33

waypoints, 85
whole-body controller, 28
wrench tasks, 46

Zero Moment Point, 28

Bibliography

F. Aghili. A unified approach for inverse and direct dynamics of constrained multibody systems based on linear projection operator: applications to control and simulation. *IEEE Transactions on Robotics*, 21(5):834–849, Oct 2005. ISSN 1552-3098. doi: 10.1109/TRO.2005.851380.

Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*, volume 26. Birkhäuser, 2012.

Chae An, C Atkeson, and J Hollerbach. Experimental determination of the effect of feedforward control on trajectory tracking errors. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 55–60. IEEE, 1986.

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

Rika Antonova, Akshara Rai, and Christopher G Atkeson. Deep kernels for optimizing locomotion controllers. *arXiv preprint arXiv:1707.09062*, 2017.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Karl Johan Åström. Theory and applications of adaptive control—a survey. *Automatica*, 19(5):471–486, 1983.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20, 1997.

John Baillieul. Kinematic programming alternatives for redundant manipulators. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 722–728. IEEE, 1985.

Dmitry Berenson, Joel Chestnutt, Siddhartha S Srinivasa, James J Kuffner, and Satoshi Kagami. Pose-constrained whole-body planning using task space region chains. In *IEEE-RAS International Conference on Humanoid Robots*, pages 181–187. IEEE, 2009a.

Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, Alvaro Collet, and James J Kuffner. Manipulation planning with workspace goal regions. In *IEEE International Conference on Robotics and Automation*, pages 618–624. IEEE, 2009b.

Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

James E Bobrow, Steven Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The international journal of robotics research*, 4(3):3–17, 1985.

M. Al Borno, M. de Lasa, and A. Hertzmann. Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics*, 19(8):1405–1414, Aug 2013. ISSN 1077-2626. doi: 10.1109/TVCG.2012.325.

K. Bouyarmane and A. Kheddar. On Weight-Prioritized Multi-Task Control of Humanoid Robots. *IEEE Transactions on Automatic Control*, in revision 2015. URL `http://www.lirmm.fr/~bouyarmane/hal-01247118.pdf`.

Karim Bouyarmane and Abderrahmane Kheddar. Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4414–4419. IEEE, 2011.

Karim Bouyarmane and Abderrahmane Kheddar. Humanoid robot locomotion and manipulation step planning. *Advanced Robotics*, 26(10):1099–1126, 2012.

Stephen Boyd and Shankar Sastry. On parameter convergence in adaptive control. *Systems & control letters*, 3(6):311–319, 1983.

Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

E. Brochu, V. M. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *ArXiv e-prints*, December 2010. URL `http://adsabs.harvard.edu/abs/2010arXiv1012.2599B`.

J. Buchli, F. Stulp, E. Theodorou, and S. Schaal. Learning variable impedance control. *The International Journal of Robotics Research*, 30(7):820–833, April 2011. ISSN 0278-3649.

Roberto Calandra, Nakul Gopalan, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian gait optimization for bipedal locomotion. In *International Conference on Learning and Intelligent Optimization*, pages 274–290. Springer, 2014.

S. Calinon and Z. Li. Statistical dynamical systems for skills acquisition in humanoids. *IEEE-RAS International Conference on Humanoid Robots*, pages 323–329, 2012.

S. Calinon, D. Bruno, and D.G. Caldwell. A task-parameterized probabilistic model with minimal intervention control. In *IEEE International Conference on Robotics and Automation*, May 2014. ISBN 978-1-4799-3685-4.

Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, 2016.

Sylvain Calinon, Irene Sardellitti, and Darwin G Caldwell. Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. In *IEEE International Conference on Intelligent Robots and Systems*, pages 249–254, Oct 2010. ISBN 978-1-4244-6674-0. doi: 10.1109/IROS.2010.5648931.

T Chettibi, HE Lehtihet, M Haddad, and S Hanchi. Minimum cost trajectory planning for industrial robots. *European Journal of Mechanics-A/Solids*, 23(4):703–715, 2004.

Stephen L Chiu. Control of redundant manipulators for task compatibility. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 1718–1724. IEEE, 1987.

Guy M. Cloutier, Alain Jutard, and Maurice Bétemps. A robot-task conformance index for the design of robotized cells. *Robotics and Autonomous Systems*, 13(4):233 – 243, 1994. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/0921-8890(94)90010-8. URL `http://www.sciencedirect.com/science/article/pii/0921889094900108`.

Marco Cognetti, Pouya Mohammadi, and Marilena Oriolo, Giuseppeand Vendittelli. Task-oriented whole-body planning for humanoids based on hybrid motion generation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4071–4076. IEEE, 2014.

Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

D.D. Cox and S. John. A statistical method for global optimization. In *Systems, Man and Cybernetics, IEEE International Conference on*, pages 1241–1246 vol.2, Oct 1992. URL `http://dx.doi.org/10.1109/ICSMC.1992.271617`.

John J. Craig, Ping Hsu, and S. Shankar Sastry. Adaptive control of mechanical manipulators. *The International Journal of Robotics Research*, 6(2):16–28, 1987. doi: 10.1177/027836498700600202. URL `https://doi.org/10.1177/027836498700600202`.

H. Dai, A. Valenzuela, and R. Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In *IEEE-RAS International Conference on Humanoid Robots*, pages 355–360, Dec 2009. doi: 10.1109/ICHR.2009.5379547.

W. Decre, R. Smits, H. Bruyninckx, and J. De Schutter. Extending itasc to support inequality constraints and non-instantaneous task specification. In *2009 IEEE International Conference on Robotics and Automation*, pages 964–971, May 2009. doi: 10.1109/ROBOT.2009.5152477.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 279–286. IEEE, 2014.

Alexander Dietrich, Christian Ott, and Alin Albu-Schäffer. An overview of null space projections for redundant, torque-controlled robots. *The International Journal of Robotics Research*, 2015.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

J. Eljaik, R. Lober, A. Hoarau, and V. Padois. OCRA - Optimization-based Controllers for Robotics Applications. *Frontiers in Robotics and AI: Humanoid Robotics*, Building the iCub Mindware: Open-source Software for Robot Intelligence and Autonomy, submitted.

Peter Englert and Marc Toussaint. Combined Optimization and Reinforcement Learning for Manipulations Skills. In *Proceedings of Robotics: Science and Systems*, 2016.

Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 292–299. IEEE, 2013.

Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7): 1006–1028, 2014.

Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G Atkeson. Optimization based full body control for the atlas robot. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 120–127. IEEE, 2014.

Glen Field and Yury Stepanenko. Iterative dynamic programming: an approach to minimum energy trajectory planning for robotic manipulators. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2755–2760. IEEE, 1996.

M. Gienger, H. Janssen, and C. Goerick. Task-oriented whole body motion for humanoid robots. In *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, pages 238–244, Dec 2005. doi: 10.1109/ICHR.2005.1573574.

Lars Grüne and Jürgen Pannek. *Nonlinear model predictive control*. Springer, 2011.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `http://proceedings.mlr.press/v48/gu16.html`.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. 2017.

N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

Kensuke Harada, Mitsuharu Morisawa, Kanako Miura, Shinichiro Nakaoka, Kiyoshi Fujiwara, Kenji Kaneko, and Shuuji Kajita. Kinodynamic gait planning for full-body humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1544–1550. IEEE, 2008.

Vincent Hayward, Laeeque Daneshmend, and Ajit Nilakantan. Trajectory generation and control for automatic manipulation. *Robotica*, 12(2):115–125, 1994.

A. Herzog, L. Righetti, F. Grimminger, P. Pastor, and S. Schaal. Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 981–988, Sept 2014. doi: 10.1109/IROS.2014.6942678.

Sang-Ho Hyon, Joshua G Hale, and Gordon Cheng. Full-body compliant human–humanoid interaction: balancing in the presence of unknown external forces. *IEEE Transactions on Robotics*, 23(5):884–898, 2007.

Aurélien Ibanez, Philippe Bidaud, and Vincent Padois. Emergence of humanoid walking behaviors from Mixed-Integer Model Predictive Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4014 – 4021, Chicago, USA, Sept 2014. doi: 10.1109/IROS.2014.6943127.

Aurélien Ibanez, Philippe Bidaud, and Vincent Padois. *Humanoid Robotics: A Reference*, chapter Optimization-based control approaches to humanoid balancing. Springer, 2017.

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–73, February 2013. doi: 10.1162/NECO_a_00393.

Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1620–1626. IEEE, 2003.

Shuuji Kajita, Mitsuharu Morisawa, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, Kiyoshi Fujiwara, and Hirohisa Hirukawa. Biped walking pattern generator allowing auxiliary zmp control. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2993–2999. IEEE, 2006.

M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644, Sept 2011. doi: 10.1109/IROS.2011.6095096.

H. Kang and F.C. Park. Motion optimization using Gaussian process dynamical models. *Multibody System Dynamics*, pages 1–19, 2014.

O. Kanoun, F. Lamiraux, P.-B. Wieber, F. Kanehiro, E. Yoshida, and J.-P. Laumond. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *IEEE International Conference on Robotics and Automation*, 2009.

Chetan Kapoor, Murat Cetin, and Delbert Tesar. Performance based redundancy resolution with multiple criteria. In *Proceedings of 1998 ASME Design Engineering Technical Conference (DETC98), Atlanta, Georgia*, 1998.

O. Khatib, L. Sentis, J. Park, and J. Warren. Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, 1(01):29–43, 2004.

Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

Donald E Kirk. *Optimal control theory: an introduction.* Courier Corporation, 2012.

Charles A. Klein and Bruce E. Blaho. Dexterity measures for the design and control of kinematically redundant manipulators. *The International Journal of Robotics Research*, 6(2):72–83, 1987. doi: 10.1177/027836498700600206. URL `http://dx.doi.org/10.1177/027836498700600206`.

Charles A Klein and Ching-Hsiang Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):245–250, 1983.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, 2013. doi: 10.1177/0278364913495721. URL `http://ijr.sagepub.com/content/32/11/1238.abstract`.

Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

Jens Kober, Betty Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 834–839. IEEE, 2008.

Jens Kober, Erhan Öztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2650, 2011.

J Koenemann, Andrea Del Prete, Yuval Tassa, E Todorov, Olivier Stasse, M Bennewitz, and Nicolas Mansard. Whole-body model-predictive control applied to the hrp-2 humanoid. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3346–3351. IEEE, 2015.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE, 2004. URL `http://dx.doi.org/10.1109/IROS.2004.1389727`.

J Zico Kolter and Andrew Y Ng. Task-space trajectories via cubic spline optimization. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1675–1682. IEEE, 2009.

Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2013.

P. Kormushev, S. Calinon, and D.G. Caldwell. Robot motor skill coordination with EM-based Reinforcement Learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3232–3237, Oct 2010.

James Kuffner, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118, 2002.

Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2589–2594. IEEE, 2014.

Tomas Kulvicius, K Ning, Minija Tamosiunaite, and F Worgotter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *Robotics, IEEE Transactions on*, 28(1):145–157, 2012.

Tobias Kunz and Mike Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Robotics: Science and Systems*, 2012.

Steven M Lavalle. Rapidly-exploring random trees: A new tool for path planning, 1998.

S. Lee. Dual redundant arm configuration optimization with task-oriented dual arm manipulability. *IEEE Transactions on Robotics and Automation*, 5(1):78–97, Feb 1989. ISSN 1042-296X. doi: 10.1109/70.88020.

S. Lee and J. M. Lee. Task-oriented dual-arm manipulability and its application to configuration optimization. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 2253–2260 vol.3, Dec 1988. doi: 10.1109/CDC.1988.194736.

Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 0(0):0278364917710318, 2017. doi: 10.1177/0278364917710318. URL `https://doi.org/10.1177/0278364917710318`.

Alain Liegeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE transactions on systems, man, and cybernetics*, 7(12):868–871, 1977.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

M Liu, R Lober, and V Padois. Whole-Body Hierarchical Motion and Force Control for Humanoid Robots. *Autonomous Robots*, Special issue on Whole-body control for Humanoid Robots:1–13, 2015a.

M. Liu, Y. Tan, and V. Padois. Generalized hierarchical control. *Autonomous Robots*, 40(1):17–31, 2015b. ISSN 1573-7527. URL `http://dx.doi.org/10.1007/s10514-015-9436-1`.

Mingxing Liu, Yang Tan, and Vincent Padois. Generalized hierarchical control. *Autonomous Robots*, 40 (1):17–31, 2016.

R. Lober, V. Padois, and O. Sigaud. Multiple task optimization using dynamical movement primitives for whole-body reactive control. In *IEEE-RAS International Conference on Humanoid Robots*, pages 193–198, Nov 2014.

R. Lober, V. Padois, and O. Sigaud. Variance modulated task prioritization in Whole-Body Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3944–3949, Sept 2015. URL `http://dx.doi.org/10.1109/IROS.2015.7353932`.

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

R. Lober, J. Eljaik, G. Nava, S. Dafarra, F. Romano, D. Pucci, S. Traversaro, F. Nori, O. Sigaud, and V. Padois. Optimizing task feasibility using model-free policy search and model-based whole-body control. In *IEEE International Conference on Robotics and Automation*, submitted.

Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. Automatic lqr tuning based on gaussian process global optimization. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 270–277. IEEE, 2016.

S. Mason, L. Righetti, and S. Schaal. Full dynamics lqr control of a humanoid robot: An experimental study on balancing and squatting. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 374–379, Nov 2014. doi: 10.1109/HUMANOIDS.2014.7041387.

René V Mayorga, Farrokh Janabi-sharifi, and Andrew KC Wong. A fast approach for the robust trajectory planning of redunant robot manipulators. *Journal of Field Robotics*, 12(2):147–161, 1995.

Xavier Merlhiot, Jérémie Le Garrec, Guillaume Saupin, and Claude Andriot. The xde mechanical kernel: Efficient and robust simulation of multibody dynamics with intermittent nonsmooth contacts. [Online]. Availible: http://www.kalisteo.fr/lsi/aucune/a-propos-de-xde, 2012.

M. Mesbahi and G. P. Papavassilopoulos. On the rank minimization problem over a positive semidefinite linear matrix inequality. *IEEE Transactions on Automatic Control*, 42(2):239–243, Feb 1997. ISSN 0018-9286. doi: 10.1109/9.554402.

M. Mistry, J. Buchli, and S. Schaal. Inverse dynamics control of floating base systems using orthogonal decomposition. In *2010 IEEE International Conference on Robotics and Automation*, pages 3406–3412, May 2010. doi: 10.1109/ROBOT.2010.5509646.

J. Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

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

M. Mühlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick. Task-level imitation learning using variance-based movement optimization. In *IEEE/RSJ International Conference on Robotics and Automation*, pages 1177–1184, May 2009. ISBN 978-1-4244-2788-8.

Albert Mukovskiy, Christian Vassallo, Maximilien Naveau, Olivier Stasse, Philippe Soueres, and Martin A Giese. Adaptive synthesis of dynamically feasible full-body movements for the humanoid robot hrp-2 by flexible combination of learned dynamic movement primitives. *Robotics and Autonomous Systems*, 91:270–283, 2017.

Thibaut Munzer, Freek Stulp, and Olivier Sigaud. Non-linear regression algorithms for motor skill acquisition: a comparison. *Proceedings JFPDA*, pages 1–16, 2014.

Yoshihiko Nakamura and Hideo Hanafusa. Optimal redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(1):32–42, 1987.

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

Michael Neunert, Farbod Farshidian, Alexander W Winkler, and Jonas Buchli. Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters*, 2(3):1502–1509, 2017.

Silvia Monica Osnaga. *Low Rank Representations of Matrices using Nuclear Norm Heuristics.* PhD thesis, Colorado State University, 2014.

A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems*, pages 2616–2624, 2013.

Diego Pardo, Michael Neunert, Alexander Winkler, Ruben Grandia, and Jonas Buchli. Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion. In *Robotics Science and Systems*, 2017.

Jaeheung Park and Oussama Khatib. Multi-link multi-contact force control for manipulators. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3613–3618. IEEE, 2005.

N. Perrin, D. Lau, and V. Padois. Effective Generation of Dynamically Balanced Locomotion with Multiple Non-coplanar Contacts. In *International Symposium on Robotics Research*, 2015.

Jan Peters and Stefan Schaal. Learning to control in operational space. *The International Journal of Robotics Research*, 27(2):197–212, 2008a.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682 – 697, 2008b. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2008.02.003. URL `http://www.sciencedirect.com/science/article/pii/S0893608008000701`. Robotics and Neuroscience.

Quang-Cuong Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics*, 30(6):1533–1540, Dec 2014. ISSN 1552-3098.

Quang-Cuong Pham, Stéphane Caron, Puttichai Lertkultanon, and Yoshihiko Nakamura. Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots. *The International Journal of Robotics Research*, page 0278364916675419, 2016.

M. Posa, S. Kuindersma, and R. Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1366–1373, May 2016. doi: 10.1109/ICRA.2016.7487270.

Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014. doi: 10.1177/0278364913506757. URL `https://doi.org/10.1177/0278364913506757`.

Jerry Pratt and Gill Pratt. Intuitive control of a planar bipedal walking robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 3, pages 2014–2021. IEEE, 1998.

A. D. Prete and N. Mansard. Robustness to joint-torque-tracking errors in task-space inverse dynamics. *IEEE Transactions on Robotics*, 32(5):1091–1105, Oct 2016. ISSN 1552-3098. doi: 10.1109/TRO.2016.2593027.

D. Pucci, F. Romano, S. Traversaro, and F. Nori. Highly dynamic balancing via force control. In *IEEE-RAS International Conference on Humanoid Robots*, pages 141–141, Nov 2016.

C. E. Rasmussen and C. Williams. *Gaussian processes for machine learning.* MIT Press, 2006. ISBN 0-262-18253-X.

Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

L. Righetti, J. Buchli, M. Mistry, and S. Schaal. Inverse dynamics control of floating-base robots with external constraints: A unified view. In *2011 IEEE International Conference on Robotics and Automation*, pages 1085–1090, May 2011. doi: 10.1109/ICRA.2011.5980156.

Ludovic Righetti, Jonas Buchli, Michael Mistry, Mrinal Kalakrishnan, and Stefan Schaal. Optimal distribution of contact forces with inverse-dynamics control. *The International Journal of Robotics Research*, 32(3):280–298, 2013. doi: 10.1177/0278364912469821. URL `https://doi.org/10.1177/0278364912469821`.

F. Romano, G. Nava, M. Azad, J. Camernik, S. Dafarra, O. Dermy, C. Latella, M. Lazzaroni, R. Lober, M. Lorenzini, D. Pucci, O. Sigaud, S. Traversaro, J. Babic, S. Ivaldi, M. Mistry, V. Padois, and F. Nori. The CoDyCo Project achievements and beyond: Towards Human Aware Whole-body Controllers for Physical Human Robot Interaction. *IEEE Robotics and Automation Letters*, Special Issue: Human Cooperative Wearable Robotic Systems, submitted.

Sébastien Rubrecht, Vincent Padois, Philippe Bidaud, and Michel De Broissia. Constraints compliant control: constraints compatibility and the displaced configuration approach. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 677–684. IEEE, 2010.

L Saab. Generating whole body movements for dynamics anthropomorphic systems under constraints. 2011. URL `http://thesesups.ups-tlse.fr/1447/`.

Layale Saab, Oscar E Ramos, François Keith, Nicolas Mansard, Philippe Soueres, and Jean-Yves Fourquet. Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362, 2013.

J. Salini, V. Padois, and P. Bidaud. Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions. In *IEEE International Conference on Robotics and Automation*, pages 1283–1290, May 2011. URL `http://dx.doi.org/10.1109/ICRA.2011.5980202`.

Joseph Salini. *Commande dynamique pour la coordination tâche/posture des humanoïdes : vers la synthèse d'activités complexes.* PhD thesis, Université Pierre & Marie Curie, 2012.

J Kenneth Salisbury and John J Craig. Articulated hands force control and kinematic issues. *The International journal of Robotics research*, 1(1):4–17, 1982.

Stefan Schaal and Christopher G Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

L. Sentis and O. Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *IEEE-RAS International Journal of Humanoid Robotics*, 2(04):505–518, 2005. URL `http://www.me.utexas.edu/~lsentis/files/ijhr-05.pdf`.

Luis Sentis, Jaeheung Park, and Oussama Khatib. Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *IEEE Transactions on robotics*, 26(3):483–501, 2010.

B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer Science & Business Media, 2008.

Olivier Sigaud, C Salaün, and Vincent Padois. On-line regression algorithms for learning mechanical models of robots: a survey. *Robotics and Autonomous Systems*, 12:1115–1129, 2011. URL `http://www.sciencedirect.com/science/article/pii/S092188901100128X`.

Jean-Jacques E. Slotine. The robust control of robot manipulators. *The International Journal of Robotics Research*, 4(2):49–64, 1985. doi: 10.1177/027836498500400205. URL `https://doi.org/10.1177/027836498500400205`.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. *ArXiv e-prints*, February 2015.

Hak Sovannara. xde-isir wiki, *http://pages.isir.upmc.fr/hak/xdewiki/*, 12 2013. URL `http://pages.isir.upmc.fr/~hak/xdewiki/doku.php?id=xde_guide`.

Jonathan Spitz, Karim Bouyarmane, Serena Ivaldi, and Jean-Baptiste Mouret. Trial-and-Error Learning of Repulsors for Humanoid QP-based Whole-Body Control. Submitted to IEEE-RAS Humanoids, July 2017. URL `https://hal.archives-ouvertes.fr/hal-01569948`.

Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *COLT*, volume 5, pages 545–560. Springer, 2005.

N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1015–1022, 2010. URL `http://arxiv.org/abs/0912.3995`.

B. J. Stephens and C. G. Atkeson. Dynamic balance force control for compliant humanoid robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1248–1255, Oct 2010. doi: 10.1109/IROS.2010.5648837.

F. Stulp and O. Sigaud. Robot Skill Learning: From Reinforcement Learning to Evolution Strategies. *Paladyn Journal of Behavioral Robotics*, 4(1):49–61, August 2013. doi: 10.2478/pjbr-2013-0003.

Freek Stulp. `DmpBbo` – a c++ library for black-box optimization of dynamical movement primitives., 2014. URL `https://github.com/stulp/dmpbbo.git`.

Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

Freek Stulp, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. Reinforcement learning of full-body humanoid motor skills. *IEEE-RAS International Conference on Humanoid Robots*, pages 405–410, Dec 2010. doi: 10.1109/ICHR.2010.5686320.

Yuval Tassa, Tom Erez, and William D Smart. Receding horizon differential dynamic programming. In *Advances in neural information processing systems*, pages 1465–1472, 2008.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

Russ Tedrake, Scott Kuindersma, Robin Deits, and Kanako Miura. A closed-form solution for real-time zmp gait generation and feedback stabilization. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 936–940. IEEE, 2015.

Emanuel Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1071–1076. IEEE, 2011.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056. ACM, 2009.

Stanislav Uryasev and Panos M Pardalos. *Stochastic optimization: algorithms and applications*, volume 54. Springer Science & Business Media, 2013.

Mike Vande Weghe, Dave Ferguson, and Siddhartha S Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *IEEE-RAS International Conference on Humanoid Robots*, pages 477–482. IEEE, 2007.

Daniel E Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 1969.

Pierre-Brice Wieber. Constrained dynamics and parametrized control in biped walking. In *International Symposium on Mathematical Theory of networks and systems*, 2000.

Pierre-Brice Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 137–142. IEEE, 2006.

Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. *Modeling and Control of Legged Robots*, pages 1203–1234. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1_48. URL `https://doi.org/10.1007/978-3-319-32552-1_48`.

Pierre-Brice Wieber, Adrien Escande, Dimitar Dimitrov, and Alexander Sherikov. Geometric and numerical aspects of redundancy. In *Geometric and Numerical Foundations of Movements*. 2017.

Holly A. Yanco, Adam Norton, Willard Ober, David Shane, Anna Skinner, and Jack Vice. Analysis of human-robot interaction at the darpa robotics challenge trials. *Journal of Field Robotics*, 32(3):420–444, 2015. ISSN 1556-4967. doi: 10.1002/rob.21568. URL `http://dx.doi.org/10.1002/rob.21568`.

Eiichi Yoshida, Igor Belousov, Claudia Esteves, and Jean-Paul Laumond. Humanoid motion planning for dynamic tasks. In *IEEE-RAS International Conference on Humanoid Robots*, pages 1–6. IEEE, 2005a.

Eiichi Yoshida, Yisheng Guan, Neo Ee Sian, Vincent Hugel, Pierre Blazevic, Abderrahmane Kheddar, and Kazuhito Yokoi. Motion planning for whole body tasks by humanoid robot. In *IEEE International Conference on Mechatronics and Automation*, volume 4, pages 1784–1789. IEEE, 2005b.

Tsuneo Yoshikawa. Analysis and control of robot manipulators with redundancy. In *Robotics research: the first international symposium*, pages 735–747. Mit Press Cambridge, MA, 1984.

Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The International Journal of Robotics Research*, 4(2):3–9, 1985a.

Tsuneo Yoshikawa. Dynamic manipulability of robot manipulators. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1033–1038. IEEE, 1985b.

Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015.