Learning and Optimization of Anticipatory Feedback Controllers for Robot Manipulation

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the registrar's office.

Thèse n. 1234 2023
présentée le 06 mars 2023
à la Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique
École polytechnique fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Hakan Girgin

acceptée sur proposition du jury :

Prof Maryam Kamgarpour, président du jury
Dr Jean-Marc Odobez, Dr Sylvain Calinon, directeurs de thèse
Prof Tamim Asfour, rapporteur
Prof Nicolas Boumal, rapporteur
Prof Ludovic Righetti, rapporteur

Lausanne, EPFL, 2023

Biggest obstacle I ever faced was
my own limited perception of myself.
— RuPaul

To my family

Acknowledgements

Despite many things said about starting a PhD thesis, this has been one of the most eye-opening experiences of my life. Apart from the knowledge and expertise that I gathered during the work in this thesis, I also had great friendships, new life perspectives and discovered a new perception of myself via personal and professional challenges. For this, I am proud of what I have accomplished by staying true to myself no matter how hard it got in this vast fancy pool of delusion.

First of all, I would like to express my gratitude towards my supervisor Dr. Sylvain Calinon for giving me the chance of pursuing a PhD degree in his lab and in my new home Switzerland. I would like to extend my gratitude to Dr. Emre Ugur for introducing me to the world of robot learning and for providing me with lots of opportunities that prepared me for this PhD thesis. I would like to thank to all my friends at Idiap, especially Florian, Thibaut, Noémie, Julius, Teguh, Cem, Tobias, Amir, Teng, Clément, Antonio, Andras for the valuable discussions and moments of coffee breaks in the workplace, and their sincere friendship outside.

I could not have undertaken this journey without my dearest friends Edanur, Mert, Osman and Ömer and I would like to thank them for their unconditional love and support throughout these years. Special thanks to my joyful friend Didem for all of our adventures together full of laughter and great coffee, and all the positivity that she brought to my life. Another special thanks to my all-around-capable friend Asli for making me feel home with the delicious food she prepared and for being there for me. I would like to extend my special thanks to Olivia for teaching me the Swiss life by sharing amazing hikes, food, and discussions. I would like to thank Emmanuel, Éloise and my godson Zacharie for being my second family in Switzerland. Thanks should also go to Ina and Mathias, who made my last years of PhD and my journey in Montreux memorable and filled with lots of fun.

I am extremely grateful to my mother Belgin, my sister Ayça and my aunt Meral to whom I dedicate this thesis. Their support, love, and efforts to keep me away from all the negative situations are what kept me strong even in the hardest times.

Martigny, April 4, 2023 Hakan Girgin

Abstract

Programming intelligent robots requires robust controllers that can achieve desired tasks while adapting to the changes in the task and the environment. In this thesis, we address the challenges in designing such adaptive and anticipatory feedback controllers in robot manipulation tasks from the perspective of two main approaches: optimization and learning. Optimization methods determine a feedback or feedforward controller that achieves tasks by using a model of the task and the dynamics of the tasks. An optimization expert is often required for tuning, modeling, and solver selection. On the other hand, learning from demonstration (LfD) is an intuitive way of programming robots by showing them demonstrations of how to achieve a task. It requires an expert who can demonstrate the task, without the need for coding or modeling.

Many existing solvers for optimization problems in robotics are not easily adaptable, difficult to implement, and/or require tuning effort. This prevents their wide adoption, benchmarking, and potential future contributions to the solvers, as well as their direct real-time applications. Furthermore, they do not fully exploit the geometric structures that we often have in robotic tasks. In the first part of the thesis, we address these challenges by proposing a projection-based first-order optimization solver for robotics problems with geometric constraints. We show that Euclidean projections onto the manifold defined by these geometric shapes can significantly improve performance even when compared to second-order methods.

The adaptive behavior of the feedback control gain matrices found by optimal control is under-exploited in robotics. However, they are known to contain important local information about the dynamics task. We extend the *system level synthesis* (SLS) framework to build novel capabilities of these gains. In particular, we show that this anticipatory feedback controller with memory can remember and act on past states, which is crucial for tasks with time correlations. We further exploit their capabilities in the real-time adaptation of the task parameters using local information from the optimization and in hierarchical optimal control using the redundancies at the planning level.

In the second part of the thesis, we address the challenges in modeling for optimization by turning our attention to learning methods. Several open questions are discussed in this thesis: 1. What to model? (or What to learn?); 2. How to execute these models on the real robot?; 3. How to demonstrate?; and 4. How many times to demonstrate? We propose two different ways of exploiting demonstrations for designing feedback controllers.

The first method uses demonstrations to warm-start and guides an optimal control problem of planar pushing which otherwise can get stuck at local optima. The second method proposes an adaptive impedance controller that can mimic the generalization and multimodality capabilities of a learned trajectory policy. We then investigate the epistemic uncertainties in such policies and provide an active learning method to refine them iteratively.

Keywords: constrained optimization, projection-based optimization, feedback control, hierarchical optimal control, learning from demonstration, active learning, learning to control

Résumé

La programmation de robots intelligents nécessite des contrôleurs robustes qui peuvent réaliser des tâches souhaitées tout en s'adaptant aux changements dans la tâche et l'environnement. Dans cette thèse, nous abordons les défis de la conception de tels contrôleurs adaptatifs et anticipatifs dans les tâches de manipulation de robots à partir de deux approches principales : l'optimisation et l'apprentissage. Les méthodes d'optimisation déterminent un contrôleur rétroactif ou prédictif qui réalise des tâches en utilisant un modèle de celles-ci et de leurs dynamiques. Un expert en optimisation est souvent requis pour l'ajustement, la modélisation et la sélection du solveur. D'un autre côté, l'apprentissage par démonstration est une façon intuitive de programmer les robots en leur montrant comment réaliser une tâche. Cette opération nécessite un expert qui peut procurer des démonstrations de la tâche sans avoir recours à des notions avancées de programmation et/ou de mathématiques.

De nombreux solveurs existants pour les problèmes d'optimisation en robotique ne sont pas facilement adaptable, difficiles à implémenter et/ou nécessitent un effort d'ajustement. Ces points rendent leurs adoptions, évaluations, applications, et une quelconque contribution par un tiers difficiles. De plus, ils n'exploitent pas complètement les structures géométriques que nous avons souvent dans les tâches de robotique. Dans la première partie de la thèse, nous abordons ces défis en proposant un solveur d'optimisation de premier ordre basé sur la projection pour les problèmes de robotique avec des contraintes géométriques. Nous montrons que les projections euclidiennes sur la variété définie par ces formes géométriques peuvent significativement améliorer les performances même lorsqu'elles sont comparées à des méthodes du deuxième ordre.

L'adaptabilité du movement par des matrices de gains obtenues par le contrôle optimal est sous-exploitée en robotique. Cependant, on sait qu'elles contiennent des informations locales importantes sur la dynamique de la tâche. Nous étendons le *system level synthesis* (SLS) pour développer de nouvelles capacités de ces gains. En particulier, nous montrons que ce contrôleur de rétroaction anticipatif avec mémoire peut se souvenir et agir sur les états passés, ce qui est crucial pour les tâches avec des corrélations temporelles. Nous exploitons encore plus leurs capacités dans l'adaptation en temps réel des paramètres de la tâche à l'aide d'informations locales provenant de l'optimisation et dans le contrôle optimal hiérarchique en utilisant les redondances au niveau de la planification.

Dans la deuxième partie de la thèse, nous abordons les défis liés à la modélisation pour l'optimisation en nous tournant vers les méthodes d'apprentissage. Plusieurs questions ouvertes sont discutées dans cette thèse : 1. Que faut-il modéliser ? (ou que faut-il apprendre ?) ; 2. Comment exécuter ces modèles sur le robot réel ? ; 3. Comment montrer une tâche ? ; et 4. Combien de fois démonstrations a-t-on besoin ? Nous proposons deux façons différentes d'exploiter les démonstrations pour concevoir des contrôleurs de rétroaction. La première méthode utilise les démonstrations pour initier et guider un problème de contrôle optimal de poussée planaire qui peut autrement se retrouver bloqué dans des optima locaux. La deuxième méthode propose un contrôleur d'impédance adaptatif qui peut imiter les capacités de généralisation et multimodalité d'une consigne apprise de trajet. Nous investiguons alors les incertitudes épistémiques dans ces consignes et fournissons une méthode d'apprentissage actif pour les affiner itérativement.

Mots-clés : optimisation sous contraintes, optimisation basée sur la projection euclidienne, contrôle rétroactif, contrôle optimal hiérarchique, apprentissage par démonstration, apprentissage actif, apprentissage pour contrôle

List of Tables

3.1 Projections onto bounded domain, affine hyperplane, quadric and second-order cone 19
3.2 Comparison of MPC with iLQR and ALSPG for planar push 31
3.3 Comparison of motion planning for obstacle avoidance for three cases . 32

6.1 Performance of DS-DDP, DC-DDP, and WS-DDP for offline programming 75
6.2 Metric-based comparison for the ball-in-box task (in Figure 6.9). 88

7.1 conditionals and priors where $\mathcal{W}(\cdot)$ and $\mathrm{Dir}(\cdot)$ correspond to Wishart and Dirichlet distributions 95

A.1 Constraints of different interaction modes 123

List of Figures

3.1 Projection view of inverse kinematics problem. These problems can be tested online with closed-loop controllers on the provided website[1]created as extensions of the toolbox Robotics Codes from Scratch[2]. 20

3.2 Comparison of iLQR and SPGOC in terms of convergence time evolution vs the number of timesteps or horizon. 27

3.3 Illustration of inverse kinematics problems solved with the proposed algorithm. 30

3.4 SPG algorithm applied to a pusher-slider system. 31

3.5 Motion planning problem in the presence of 4 scaled and rotated rectangular obstacles. 33

3.6 MPC setup for tracking an object subject to box constraints. 34

3.7 Error in the objective and the squared norm of the constraint value during 1 min execution of MPC on Franka Emika robot. 34

4.1 (a) The robot decides where to place the object according to the initial position and with the anticipation to pick up the sugar cube and put it inside the cup. (b) Pick-and-place task with the Franka Emika robot deciding autonomously to grasp the object from different parts, as an adaptation to different initial configurations. The memory feedback property allows the robot to remember where it placed the mug and grasped the object, so that it can place it accordingly. 36

4.2 Simulated task where the robot, initially holding a coffee mug, needs to place it on the table within a certain range (brown disk), pick up a sugar cube (white cube) and drop it onto the mug. 45

4.3 Adaptation of the robot to different initial configurations shown in transparent and the grasping locations shown in solid colors. 46

4.4 Testing of the reactivity of the controller to different physical perturbations, each resulting in successful completion of the task by adaption. The plot of z-axis position (m) in time (s) gathered from the robot perturbed before grasping the object. The curves of color green, orange and blue correspond to the first, second and third screenshots respectively, while the dashed black line corresponds to the nominal solution. The robot decides on-the-fly to grasp from different locations and remembers these locations to place the yellow object without colliding with the environment. 46

4.5 (a) Nominal behavior of the robot without any adaptation. (b) The final relative height is changed at the initial time and the robot adapts to this new height with a new plan, by remembering where it grasped the object. (c) Online adaptation by increasing the relative height of the final position after grasping the object. (d) Adaptation to change of the relative height at the beginning of the movement combined with a change in the final position introduced after the grasping phase. 47

4.6 Here, we show three phases of the movement, pick, handover and place to show three different adapted behaviors, denoted as (a), (b) and (c) for each phase. The grasping locations are highlighted by a star sign placed at the same exact locations of the objects to differentiate them. Before the *pick* phase, in (a) and (c), the robot is perturbed upwards and downwards respectively, while (b) is not perturbed. After the *pick* phase, and before *handover*, a new target location to place the object which are denoted by transparent yellow cylinder is introduced, with the red ones being the original targets. Finally, in the *place* phase, the second robot can successfully adapt to new target location thanks to the proposed online adaptation scheme. It also could correctly place the object without hitting the floor by remembering where it was grasped by the first arm. 51

5.1 *From left to right:* kinematic/mechanical redundancy (morphology level), task redundancy (spatial level), planning redundancy (spatiotemporal level). 53

5.2 LQR with initially at the position shown by the cross and (a) 2 intermediary tasks at time step t with hierarchies $V_1^t > V_2^t$ and the final task *Goal*, (b) 2 intermediary tasks at time step t, 1 intermediary task at time step $t+1$ with hierarchies $V_1^t > V_2^t > V_3^{t+1}$, and the final task *Goal*, (c) 2 intermediary tasks at time step t, 2 intermediary tasks at time step $t+1$ with hierarchies $V_1^t > V_2^t > V_3^{t+1} > V_4^{t+1}$, and the final task *Goal*. Tasks are represented with Gaussian ellipsoids, while the resulting trajectory is represented by black lines. The grey dashed line represents the shortest line between the mean of the Gaussian ellipsoid of V_2^t and the line ellipsoid (thin Gaussian ellipsoid) V_1^t. 58

5.3 Comparison between nullspace (solid black line) and standard formulation (dashed lines) with scaling δ_1, δ_2, δ_3 and δ_4 are 1, 0.1, 0.01 and 0.001 respectively, using a point mass object with double integrator dynamics. The agent has to go to the goal point G_T as its primary task, at the final time step T, with variance shown by a green circle. If possible, it should pass through a viapoint V_{T-k}, as a secondary task, at the time step $T-k$, with the same variance shown by orange circle. When the secondary task is far away from the primary task in space and time, nullspace planning tries to pass through V_{T-k}, and still succeed to accomplish the primary task. On the other hand, no matter the scaling of the cost, standard control cannot have the same performance. 59

5.4 Two agents (black and red) have a primary task of reaching their goal positions at time step T and a secondary goal to meet at some position at time step $T/2$. If obstacles perturb their executions, we observe the nullspace effect with a shift of the meeting position. 60

5.5 A robotic application of nullspace structure within LQR. 61

5.6 10 realizations (starting from different initial positions) of the primary controller and the nullspace controller 65

6.1 Convergence curve 75

6.2 Solving time 75

6.3 Tracking performance under disturbance. The dashed lines present the tolerance. 76

6.4 Pushing task with face switching. The manipulator starts from (a) and selects an optimal face (orange) and contact point (b) for pushing, until reaching the planned face switching point (c). Next, the manipulator changes to face (d) and touches the object again at the selected point (e), followed by the next phase of pushing (f), until reaching the final target (g). This example is for $N_s = 1$. (d)$\sim$(f) should be repeated if $N_s > 1$. The colored line is used to express the current active face, and the black arrow in (d) represents the face switching process. 77

6.5 Planar pushing with face switching. Both simulation and experimental results reach final targets within tolerance. 78

6.6 Drawing task for a 2D point mass. The parameters of the proposed controller are set to $\boldsymbol{\Sigma}_{\boldsymbol{q}}=\sigma_{\boldsymbol{q}}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}=\sigma_{\dot{\boldsymbol{q}}}\boldsymbol{I}$, $\boldsymbol{K}=500\boldsymbol{I}$ and $\boldsymbol{D}=3\sqrt{500}\boldsymbol{I}$. The original controller and the proposed controller produce the same behavior if the tolerance of tracking the probabilistic model is set close to zero (left plot). For the tolerances $\sigma_{\boldsymbol{q}}=10^{-5}$ and $\sigma_{\dot{\boldsymbol{q}}}=10^{-3}$, the proposed controller adds stabilizing feedback terms. This results in a probabilistic feedback control that is robust to perturbations (right plot). 82

6.7 Multiple-mode navigation task for a 2D point mass. The parameters of the proposed controller are set to $\boldsymbol{\Sigma}_{\boldsymbol{q}}$=$10^{-5}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}$=$10^{-3}\boldsymbol{I}$, $\boldsymbol{K}$=$100\boldsymbol{I}$ and $\boldsymbol{D}$=$3\sqrt{100}\boldsymbol{I}$. The dashed trajectories show the behavior of the original controller (red) and the proposed controller (blue) without perturbations. The solid trajectories show the result of all controllers under the impact of an external force $\boldsymbol{f}_{\text{ext}}$ that pushes the point mass in positive q_2-direction for 0.1 seconds. 84

6.8 Target reaching task for a 2D point mass initialized at three different positions depicted by the small black circles. The parameters of the proposed controller are set to $\boldsymbol{\Sigma}_{\boldsymbol{q}}$=$10^{-5}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}$=$10^{-3}\boldsymbol{I}$, $\boldsymbol{K}$=$100\boldsymbol{I}$ and $\boldsymbol{D}$=$3\sqrt{100}\boldsymbol{I}$. The target, depicted by the filled black circle, jumps from the position indicated by the grey circle in the lower right corner to the final target in the upper right corner after 2.3 seconds. The duration of the motion is 4 seconds. 86

6.9 The Franka Emika robot has to drop a ball into the box using the presented imitation controller (blue). The skill is demonstrated (orange trajectories) for a given box position (orange crosses). Experiment 1 (*left*): The original controller (red) and the proposed controller are tested on two different static box positions (context 1 and 2). Experiment 2 (*center* and *right*): The proposed controller is tested in a dynamic scenario, where the context variable changes during the execution (i.e. the box is moving from position 1 to 3) such that the position reference (green) changes accordingly. The proposed controller adapts its stiffness based on the variability of the demonstrations and based on the correlation of the trajectory phase with the context variable (indicated by the stiffness ratio). 87

7.1 *(a)* Demonstrations and *(b)-(c)* reproductions of a reaching task in a cluttered environment. The goal position is denoted by G and the obstacles are represented as dashed rectangles. The demonstrated trajectories are depicted with red lines. The policy samples acquired from the BGMM and PoE are depicted by colored lines. 97

7.2 Uncertainty colormaps of the learned control policy for a reaching task in a cluttered environment. *(a)*, *(b)* and *(c)* show the total, aleatoric and epistemic uncertainties of the BGMM, respectively. High to low uncertainties are depicted by colors ranging from yellow to purple. *(d)* depicts the information-density cost and the Gaussian components of the GMM model approximating this cost. 100

7.3 Evolution of the quadratic Rényi entropy of (a) the GMM model that approximates highly uncertain regions and (b) the marginal BGMM model. Top figures represent the evolution for the proposed active learning, while the error bars in bottom figures show the mean and the standard deviation of 5 different random exploration for 5 iterations. 103

7.4 Reproductions of the learned policy after 10 iterations of active learning. The numbers on (a) denotes the location the query point at each iteration of active learning. 103
7.5 The task is to put the cup inside a box covered from top and bottom, starting from anywhere in the space. The robot has to maintain a specific end-effector orientation to perform the task without pouring the cup. The main challenge is not to collide with the box and the other obstacles in the environment. 104
7.6 Overview of the simulated pouring environment. 107
7.7 Subset of demonstrations for different contexts. 108
7.8 Quantitative results for simulated 1D context pouring. 108
7.9 Quantitative results for simulated 2D context pouring. 110
7.10 Quantitative results for simulated 3D context pouring. 111
7.11 Visualization of the context space during the first 3 iterations of the active learning process. The heatmap represents the entropy of the epistemic uncertainty, yellow indicating high uncertainty. Demonstrations are shown as grey stars. The context chosen for the next demonstration is shown as a red star. Transparent ellipses show the marginal distribution of the ProMP in the context space. 112
7.12 Overview of the pouring task with a 7-axis robot. 112

8.1 Non-robust linear quadratic tracking controller (LQT-ADMM) and robust system level synthesis controller (SLS-ADMM) executed from 5 different initial positions. SLS-ADMM is optimized to be robust with respect to the control bounds. 118

A.1 Kinematics of pusher-slider system allowing face switching. 124

Contents

Acknowledgements **i**

Abstract (English/Français) **iii**

1 Introduction **1**
- 1.1 Motivation & Challenges 4
- 1.2 Organization of the thesis 5

2 Background **7**
- 2.1 Behavior Primitives 7
  - 2.1.1 Learning Primitives from Demonstration 7
  - 2.1.2 Learning policies for control 8
  - 2.1.3 Improvement of Primitives using Active Learning 10
- 2.2 Optimal Control 11
  - 2.2.1 Linear quadratic tracking with least-squares 12
  - 2.2.2 Batch-LQT 12
- 2.3 Robot Dynamics 13

I Optimization of Controllers **15**

3 Projection-based first-order constrained optimization solver for robotics **17**
- 3.1 Related work 18
- 3.2 Background 19
  - 3.2.1 Euclidean projections onto sets 19
  - 3.2.2 Projection view of inverse kinematics 20
- 3.3 Augmented Lagrangian Spectral Projected Gradient Descent for Robotics 21
  - 3.3.1 Spectral projected gradient descent 21
  - 3.3.2 Augmented Lagrangian spectral projected gradient descent (ALSPG) 22
  - 3.3.3 Handling of inequality constraints 24
  - 3.3.4 Optimal Control with ALSPG 25
- 3.4 Convex polytope projections and linear transformations 26
  - 3.4.1 Convex polytope projections 27
  - 3.4.2 Linear transformation of projections 28

3.5 Experiments 29
3.5.1 Constrained inverse kinematics 29
3.5.2 Motion planning and MPC on planar push 30
3.5.3 Motion planning with obstacle avoidance 32
3.5.4 MPC on Franka Emika 32
3.6 Conclusion 33

4 Robust Anticipatory Robot Skills with Memory 35
4.1 Related Work 37
4.2 Background 38
4.3 Methods 40
4.3.1 Extended system level synthesis (eSLS) 40
4.3.2 Iterative system level synthesis (iSLS) 41
4.4 Experiments and Results 44
4.4.1 Simulated task 44
4.4.2 Pick-and-place task 46
4.4.3 Bimanual handover task 49
4.5 Conclusion 50

5 Nullspace Methods in Planning and Control 53
5.1 Inverse kinematics with nullspace structure 53
5.2 Nullspace structure in linear quadratic tracking (LQT) 56
5.3 Experiments 57
5.3.1 Proof-of-concept examples 57
5.3.2 Robot Simulation 59
5.4 Nullspace feedback controller in system level synthesis (SLS) 60
5.5 Bilevel optimization of hierarchical optimal control 63
5.6 Experiments 64
5.6.1 Viapoint reaching 64
5.7 Conclusion 66

II Learning of Controllers 67

6 Learning robotic skills from demonstrations 69
6.1 Learning trajectory models to warm-start optimal control 70
6.1.1 Problem formulation 71
6.1.2 Demonstration-started DDP (DS-DDP) 72
6.1.3 Demonstration-constrained DDP (DC-DDP) 73
6.1.4 Warm-starting DDP (WS-DDP) 73
6.1.5 Adaptation to disturbance 73
6.1.6 Experiments 74
6.1.7 Conclusion 78

6.2 Learning trajectory models for adaptive control 79
6.2.1 Probabilistic Movement Primitives 79
6.2.2 Formulation of the proposed controller 80
6.2.3 Single Mode 82
6.2.4 Multiple Modes 83
6.2.5 Feedback of the Context 85
6.2.6 Experimental Validation 87
6.2.7 Conclusion 89

7 Active learning of feedback controllers 91
7.1 Related Work 93
7.2 Background 94
7.2.1 Bayesian Gaussian Mixture Model 95
7.2.2 Product of Experts 96
7.3 Active Learning of Control Policies 97
7.3.1 Uncertainty decomposition 98
7.3.2 Rényi entropy of the posterior distribution 99
7.3.3 Information-density cost for active learning 100
7.3.4 Experiments 101
7.4 Active Learning of Trajectory Policies 105
7.4.1 Experiments 106
7.4.2 Simulated pouring 107
7.4.3 Real robot pouring task 111
7.5 Conclusion 112

8 Discussion & Future Work 115
8.1 Projection-based optimization for robotics 115
8.2 System level synthesis: perspectives on robust optimal control and inverse optimal control 116
8.3 Combining learning and optimization of controllers 118

9 Conclusion 121

A Appendix 123
A.1 Planar Push Dynamics 123
A.1.1 Kinematics 123
A.1.2 Generalized Motion Cone 123
A.1.3 Generalized Motion Equation 124
A.2 Mathematical background 125
A.2.1 Pseudo-inverse and nullspace of a row matrix 125
A.2.2 Properties of the nullspace matrix 126
A.2.3 Separable costs and constraints 126

A.2.4 Expectation of some linear and quadratic forms 127
A.2.5 Probabilistic inequalities 127
A.2.6 Projection onto sublevel set of a convex function 128
A.3 Controller derivation 129
A.3.1 eSLS closed-loop map 129
A.4 Nullspace in inverse kinematics and optimal control 131
A.4.1 Nullspace method for linear equality constrained QP 131
A.4.2 Bilevel optimization 131

Bibliography **133**

Curriculum Vitae **147**

1 Introduction

There are many layers to the programming of robots to make them achieve a desired task. Robots are merely composed of links whose positions and velocities need to be controlled by the joints that connect them. Depending on the robotic platform used, these joints (or motors at the joints) can be controlled by position, velocity, torque or current control commands with an ordering from the highest level to lowest level control. *Control* field offers ways to design open-loop and closed-loop controllers knowing the state of the robot and the control actions that can be applied to the joints.

Open-loop control consists of determining a reference trajectory of states and control commands to track, and applying these commands blindly, without any feedback of the current situation. In contrast, *closed-loop control* is when these reference trajectories and control commands are used in a way to correct themselves using a sensory feedback of the environment and/or of the robot itself. Designing such controllers requires a careful tuning and/or optimization of either the trajectory parameters or the control parameters. Optimization and learning of these parameters has become increasingly popular with the advances of the computational power of devices.

For example, in robotics, the problem of inverse kinematics has been thoroughly investigated and algorithms have been created based on the Jacobian matrix of the forward kinematics function [1]. These algorithms are based on the optimization of an objective function of the joint configurations, that encodes reaching a desired end-effector pose and finding the optimal joint configuration. Taking this as a reference configuration to reach, either an offline path can be planned starting from an initial configuration, or feedback controllers (e.g. a proportional-derivative controllers) can be designed to drive the robot to the reference configuration. Planning this path or determining the optimal feedback controllers via optimization is the topic of *optimal control* (or *trajectory optimization* or *motion planning*).

Iterative linear quadratic regulator (iLQR) [2] and differential dynamic programming (DDP) [3] are efficient dynamic programming-based algorithms to solve optimal control problems. An important property of these algorithms is that they do not only output open-loop control commands or trajectories, but they also find a locally stable optimal feedback controller with time-variant gain matrices. However, in robotics, these feedback matrices have been under exploited as the feedback mechanism in model predictive control (MPC) has gained more attraction. Even though, one can design various ways of exploiting these matrices in conjunction with MPC for better performances and adaptation [4], their properties and capabilities are still an open research question.

The methods that output feedback controllers are, in their basic forms, unconstrained optimization solvers. However, oftentimes, we need to enforce constraints such as joint and torque limits, obstacle avoidance, virtual safety fixtures or force closures to optimize trajectories that take the physical real life constraints into account. There are many commercially available second-order solvers to address general constrained optimization problems such as SNOPT [5], SLSQP [6], LANCELOT [7] and IPOPT [8]. However, in robotics, the literature on optimization is still focusing on developing solvers that effectively solve each robotic problem separately. For example, in the motion planning literature, one can find many constrained variants of iLQR and DDP [9, 10, 11], as well as trajectory optimization solvers such as TrajOpt [12], CHOMP [13]. The reason for such diversity and abundance of optimizers in robotics is mainly the fact that each task or each problem in robotics can be described in such a unique way that we can not assume that one solver is the most efficient for all problems. Also, the convergence and the speed of the optimizers depend significantly on the initialization, the hyperparameter selection and the mathematical modeling of the problem. This requires a great expert knowledge in coding, tuning and modeling optimization problems, even though a commercial solver is used.

A closer look at the modeling of the constrained optimization problems in robotics reveals that the constraints are often described by a *geometric primitive* or a combination of such primitives. Examples include reaching a point with an accuracy represented by an ellipsoid, avoiding obstacles represented as convex polytopes, exploiting semi-positive definite ellipsoids of manipulability and stiffness matrices and constructing virtual fixtures as hyperplanes. A common property of most of these primitives is that Euclidean projections onto the manifold defined by them can be computed efficiently. Also, projection-based optimization algorithms in these cases have been shown to perform better than plain constrained version of the same problems [14].

Another affordable and intuitive solution to the difficulty in optimization in robotics, especially in modeling of objectives and the task dynamics, comes from the field of *learning*, in particular, *learning from demonstration* (LfD). LfD requires an expert with the knowledge on how to achieve a task (e.g. factory employee on the assembly line) and not with the engineering knowledge to program the robot, in contrast to the optimization experts mentioned before. This expert shows a demonstration of the task either by grabbing the robot arm (i.e. kinesthetic teaching), teleoperating using haptic devices, or just letting the robot "look" at them (i.e. learning from observation). Indeed, the robot records all the sensory information in the form of datasets to later learn from them. Learning here refers to encoding the demonstration data into compact (probabilistic/deterministic/(un)parametric) models that can achieve the task.

The general definition of LfD raises naturally several questions that have been investigated as challenges of LfD in robotics: 1. What to model? (or What to learn?); 2. How to execute these models on the real robot?; 3. How to demonstrate?; and 4. How many times to demonstrate?. Depending on the task, one may choose different sources of sensory information to model (e.g. joint positions and time, end-effector position and force etc.) in the form of *movement primitives*. These primitives can be combined in parallel and in series to form more complex behaviors. The choice of the model of the movement primitive affects directly the control law that is going to execute the output of the learned model. For example, if a primitive model outputs a trajectory to track, an impedance controller can be designed; whereas if the output is already joint torque commands, then a different procedure should be followed as in [15]. Several combinations of movement primitives with a control strategy are present in the literature [16].

An interesting note here is that unlike many other application fields of *machine learning* (e.g. computer vision), in robotics, we do not have access to many data, as collecting demonstrations from experts is costly. We can not expect to be efficient to demonstrate a movement in 100 random ways until making sure that the robot learned how to achieve the task. One can find very important to learn robust generalizable models from as few demonstrations as possible. This brings back Questions 3 and 4 into discussion.

Consider the case when we have a model of the movement learned from one random demonstration of the task and we measure the quality (generalizability, robustness, adaptation etc) of the model and decide to demonstrate once more. How one can decide how different demonstration they should do so that the robot learns the most. If the demonstration is too close, then it is basically the same information that does not change the model much. On the other hand, if the demonstration is too far, then the model can learn just an average model that performs very bad everywhere. Also, when do they decide that the robot learned enough.

Active learning is a sub-field that addresses these questions in principled manners. It uses the uncertainty in the probabilistic models to define the input space regions where the model is the most uncertain. It then selects the next demonstrations from there in order to maximize the information gain [17]. The type of uncertainty caused by the lack of data in the learning of the model is called the *epistemic uncertainty* as opposed the *aleatoric uncertainty* representing variations and noise in the model. In robotics, active learning has been explored with methods based on heuristics, and not the *epistemic uncertainties* in the model [18, 19].

With the view of robotic problems explored from the perspective of optimization and learning, there are several open research questions and under exploited areas that we cited until now. In the next section, we will detail these areas with their corresponding challenges that motivated this thesis and its accompanied contributions.

1.1 Motivation & Challenges

Many existing second-order solvers are not easily adaptable and/or difficult to implement, which hinders benchmarking and potential improvements. As powerful as these solvers may be, their applications for finding real-time fast feedback mechanisms such as closed-loop inverse kinematics and MPC requires tuning and adaptation of the solver. Furthermore, the fact that the constraints in robotics can often be described using a geometric shape (e.g. friction cones, bounding boxes or reaching precisions as spheres) has not been exploited in these solvers. These constraints have in common that they can be formulated as easy-to-compute projections rather than constraints. In Chapter 3, we address these challenges by proposing a very simple, yet powerful first-order solver that can be easily implemented without having large memory requirements. This chapter argues that exploiting the projection capability of these sets instead of treating them as generic constraints in the solvers significantly improves the performance.

A fast constrained optimization solver as the one presented in Chapter 3 can be used as a path planner that outputs optimal trajectories to a separate controller, or as a short horizon optimal controller to be used with MPC feedback mechanism. In DDP and iLQR, we can also obtain time-variant local optimal feedback controllers with feedback gain matrices and feedforward control commands that can be exploited either directly in case of small disturbances, or in MPC to be re-optimized. In robotics, the capabilities of these feedback and feedforward control outputs of such solvers in terms of adaptation are underexploited. Although, feedback gain matrices have been shown to contain local information about how to achieve a task [20]. In Chapter 4, we address this challenge by proposing a new framework based on system level synthesis (SLS) [21] and showing that adaptation capabilities of these terms without having to resort to replanning strategies such as MPC. In Chapter 5, we extend these capabilities in hierarchical optimal control exploiting redundancies in the planning (e.g. sparse cost function). This way, we propose a principled way of solving hierarchical optimal control problems by also providing associated feedback gains and show their fast local adaptation in the case of a change in the desired states.

Chapters 3 to 5 assume that we have access to a perfect model of the dynamics of the robot or of the task, and we have the knowledge of designing a cost function that achieves the task at hand. Even though, in practice, we can create trajectories and controllers that achieve the task at hand, it is rather difficult to assume that they can be generalized to different variations of the same tasks or of the environment. Also, programming robots by modeling and coding every aspect of the task may be time-consuming and requires expertise in robotics.

A standard way is to learn trajectory policies and then combine them with a controller. A challenge here is to design such controllers that exploit the generalization capabilities of the learned trajectory policy as much as possible (Section 6.2). But when the task dynamics need to be modeled, it becomes more complicated to find such controller. In this case, we prefer optimal control strategies, and in particular the feedback mechanism in MPC. There is not a principled way of exploiting learned trajectory policies in MPC. In Section 6.1, we show a way to combine learned trajectory policies to warm-start optimal control, preventing it to being stuck at local minima.

Learning policies and dynamics models from data brings its own challenges inherent to data science. As gathering demonstrations is costly in LfD, a challenge is to reduce the number of demonstrations and quantify the epistemic uncertainties in the probabilistic models that represent the policies. In Chapter 7, we provide principled ways of measuring what constitutes a good demonstration for trajectory and control policies.

1.2 Organization of the thesis

Chapter 2 gives the general background to the thesis with related work in the literature. Chapter 3 to Chapter 7 are divided into two parts: optimization of controllers and learning of controllers.

Part I: Optimization of Controllers: Three chapters, from Chapter 3 to Chapter 5 are presented in this part. Chapter 3 presents a projection-based first-order constrained optimization solver for robotics problems with geometric constraints. Chapter 4 investigates a method to optimize feedback controllers as opposed to open-loop controllers as in the previous section. This method serves as a basis to create robust anticipatory robot skills that can be adapted fast online to the changes in the environment. We further exploit such controllers in the problems of hierarchical motion planning in Chapter 5 and propose a principled way of finding adaptive feedback controllers using nullspace methods.

Part II: Learning of Controllers: This part is composed of two chapters: Chapter 6 and Chapter 7. Chapter 6 gives two contributions for learning robotic skills from demonstration. We first present a way to exploit expert demonstrations to warm-start and guide optimal control problems in a challenging planar pushing task. Then, we present our contribution on how to design an adaptive and generalizable impedance controller that mimics expert demonstrations. In order to iteratively improve the learned skills, Chapter 7 investigates what constitutes a good demonstration and how to give demonstrations in the context of active learning using epistemic uncertainties of the dataset. It presents two contributions with active learning on control policies and trajectory policies, and discusses their comparisons.

Chapter 8 gives discussion on the contributions of this thesis and the relevant future work. Chapter 9 concludes the thesis.

2 Background

2.1 Behavior Primitives

In robotics, “movement primitives” or “behavior primitives” are often used to describe the motor skills that can generalize over similar tasks and that can be organized in series (succeeding movements) and in parallel (hierarchical movements) to form a library of unit movements. I will use the term behavior primitives in this thesis proposal to enforce the idea that the primitives used in human-robot collaboration tasks require not only movements but also applying forces without movements as well. A behavior primitive is a policy that can be described by three levels of abstractions: task-level, trajectory-level and action-state level. In task-level abstraction, a behavior primitive is a function $\pi : \boldsymbol{x}, \boldsymbol{s} \rightarrow \boldsymbol{o}$ that maps a sequence of states $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_T]$ and contexts $\boldsymbol{s}$ to a sequence of options $\boldsymbol{o} = [o_1, ..., o_T]$, i.e. discrete set of predefined action units [22]. Trajectory-level representation function $\pi : \boldsymbol{x}_0, \boldsymbol{s} \rightarrow \boldsymbol{\tau}$ takes an initial state $\boldsymbol{x}_0$ and contexts $\boldsymbol{s}$ to a trajectory $\boldsymbol{\tau}$. These two types can sometimes be referred to as *control policy* and *trajectory policy* in the learning literature or *feedback control* and *open-loop trajectory* in the control literature, respectively. Finally, action-state level abstraction is a function $\pi : \boldsymbol{x}_t, \boldsymbol{s} \rightarrow \boldsymbol{u}_t$ mapping the current state $\boldsymbol{x}_t$ and contexts $\boldsymbol{s}$ to the current control command (or action) $\boldsymbol{u}_t$ [23].

2.1.1 Learning Primitives from Demonstration

Learning skills from demonstration requires an expert to show how to perform a given task. Teaching can be done by recording observations of human movements using visual input/wearable devices or measurements of robot motion using kinesthetic teaching, i.e., guiding the robot arm to follow a desired trajectory by holding it. While the former introduces difficulties such as correspondence problem, that is to find a mapping from the human movement to the robot joint motion, the latter is more widely used in robotics because it eliminates this problem [24].

Learning from demonstration (LfD) can be divided into two broad categories: behavior bloning (BC) and inverse reinforcement learning (IRL), also called as inverse optimal control (IOC). BC consists of inferring the parameters of a behavior primitive model from a given dataset of demonstration by minimizing a predetermined cost function. When this cost function is not available beforehand, it can be learned via IRL methods [25, 26, 27, 28, 29].

As policy gradient methods in reinforcement learning, BC and IRL methods can also be divided into model-based and model-free. Most of the behavior primitive learning from demonstration works belong to the model-free behavioral cloning [30, 31, 32, 33, 34, 35, 36]. Recently proposed Bayesian Gaussian mixture models (BGMMs) [37] learn a joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in action-state abstraction, then condition on the current state $\boldsymbol{x}_t$ to find the conditional (predictive) distribution of the policy $p(\boldsymbol{u}_t|\boldsymbol{x}_t)$. Stability and hierarchical organization are also provided within product of experts (PoE) [38] framework. Model-free methods assume that the reproduced trajectories are feasible within the dynamics and kinematics constraints of the robot and of the environment. However, in manipulation tasks, the system robot-environment to be controlled becomes immediately underactuated [39], even though the robot actuation is redundant.

Model-based methods refer to a method of policy learning using the knowledge of the robot and environment dynamics. Learning dynamics model is beneficial as learning policy in BC and IRL methods becomes data-efficient and fast. The inferred policy can also be easily transferred across different tasks within the same environment since it satisfies the system dynamics. Model-based approaches such as [40, 41] propose to match the distribution of the trajectory created from rollouts of the dynamics model and the policy to the distribution of the trajectory induced by expert demonstration. These methods require the observation of control inputs in the demonstration dataset. This is not always accessible in kinesthetic teaching of interaction tasks since the interaction torques applied by the human would bias the torques measured on the joints of the robot. Learning from observation [42], a relatively new area of LfD, offers a solution to this problem. To infer the unobserved actions, it uses an inverse dynamics model which has been previously learned using an exploratory policy. These actions are then exploited for the learning of a policy [43].

2.1.2 Learning policies for control

In LfD, there are different ways of modeling the demonstration data in the form of control policies or trajectory policies. These policies are often modeled as (unnormalized) probability density functions as exploiting the uncertainties of the learning method and the variations in the demonstrations are crucial for the success of the task (see Chapter 7). Moreover, tractable probabilistic models offer important characteristics such as conditioning, marginalization, products and weighted summation (for mixture densities) that proved to be very useful in robotics literature. These operations allowed models to have task-(in)dependent skills, to be combined in parallel for fuse of information from different types of demonstrations, and to represent multimodal skills to encode different ways of performing the same task [15].

The control policies defined as probability distributions $p(\boldsymbol{u}_t|\boldsymbol{x}_t)$ and the trajectory policies defined as probability distributions $p(\boldsymbol{\tau})$ are related by the equation

$$p(\boldsymbol{\tau}|\boldsymbol{s}) = p(\boldsymbol{x}_0)\prod_{t=1}^{T} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{s})$$

where

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{s}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)p(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{s})d\boldsymbol{u}_t.$$

where $\boldsymbol{s}$ is the context variables that depend on the task. Learning distributions with respect to this variable allows models to generalize to different tasks parameters such as geometric aspects [44], mass of the objects [45] etc.

The advantages of learning trajectory policies compared to control policies are: 1. ease of collecting demonstrations for learning trajectory policies as there is no need to record control commands (which sometimes cannot be done); and 2. better generalization capabilities in terms of the trajectory in a task with simple dynamics. The disadvantages are: 1. non-dynamics-aware learning process hinders good generalization capabilities in the presence of complex dynamics; and 2. it needs to be combined with a separate controller which may not reflect the full capabilities of the learned model.

There are several ways of learning trajectory policies from demonstration. The most common way is to collect demonstrations $\boldsymbol{\mathcal{D}} = \left\{\{\boldsymbol{q}_t^k, \dot{\boldsymbol{q}}_t^k\}_{t=1}^T, \boldsymbol{s}_k\right\}_{k=1}^K$ where K denotes the number of demonstrations from the robot via kinesthetic teaching. Each demonstration k contains the robot's position and velocity trajectories $\{\boldsymbol{q}_t^k, \dot{\boldsymbol{q}}_t^k\}_{t=1}^T$ with a time horizon T, and some optional context variables $\boldsymbol{s}_k$ describing the information that affects the whole trajectory, such as environment properties (e.g. size of objects). We then select a parametric (or nonparametric) model $p(\boldsymbol{\tau}|\boldsymbol{s})$ [32] or $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{s})$ [35] and learn its parameters (or hyperparameters) that maximizes this model likelihood. Inputting reference trajectories into a stable controller is a standard way of designing such controllers.

Learning control policies requires a recording of control commands on top of the demonstrations required for learning the trajectory policies, i.e. $\boldsymbol{\mathcal{D}} = \left\{\{\boldsymbol{q}_t^k, \dot{\boldsymbol{q}}_t^k, \boldsymbol{u}_t^k\}_{t=1}^T, \boldsymbol{s}_k\right\}_{k=1}^K$. With these demonstrations, one can learn a parametric (or nonparametric) state-dependent model $p(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{s})$ [33, 37]. Learning control policies has been proved to be useful in the reinforcement learning literature [46], with the challenge of the stability of the controllers. Indeed, learning probabilistic models of feedback controllers without any guarantees still requires some kind of local stability criterion for safe application in the robot. In [47], we describe a way to learn locally stable control policies that can generalize well to different tasks using generative adversarial networks (GANs). Another such example can also be found in Section 7.2.2.

2.1.3 Improvement of Primitives using Active Learning

A collection of work focuses on improving and fine-tuning learned movement primitive representations using reinforcement learning (RL) [48, 49] and iterative learning control (ILC) [50]. LfD provides a good initial start to these methods, that will then improve it using a reward function and random explorations. In contrast, information-theoretic explorations in model-free BC methods to enhance the quality and the generalizability of the learned behavior primitive have been exploited only in very few works [18, 51, 19].

An active learning framework develops and tests new hypotheses in an interactive learning process, in contrast to passive learning systems that attempt to explain the model according to the available training data. Consider an example of policy representation with action-state level abstraction where the robot learns a probabilistic model $p(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{s})$ given N context variables. To improve this policy, the robot is expected to request a demonstration for the $(N+1)$-th context variable using the uncertainty in the posterior model $p(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s})$ which is assumed to be available, for example, using variational inference methods. One of the simplest and widely used active learning method is uncertainty sampling. Using an uncertainty measure, the learner is expected to request a query point from the input space where it is the most uncertain about. An example of uncertainty measure, entropy, can be maximized as

$$\boldsymbol{s}_H^* = \operatorname*{arg\,max}_{\boldsymbol{s}} - \int_{\boldsymbol{u}_t, \boldsymbol{x}_t} p(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s}) \log p(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s}) d\boldsymbol{u}_t d\boldsymbol{x}_t, \tag{2.1}$$

to provide an information-theoretic approach to determine the optimal context variable to query next. The learned model $p(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s})$ incorporates both aleatoric (caused by the variance in the demonstration) and epistemic uncertainties (caused by never-seen data space and lack of sufficient data). We are interested in minimizing the epistemic uncertainties in our model. To this end, we can train several probabilistic representations with different local convergence properties $\{p_e(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{s})\}_{e=1}^E$ where E represents the number of models learned. We can maximize the disagreement between each individual model and the consensus model

$$\boldsymbol{s}_E^* = \operatorname*{arg\,max}_{\boldsymbol{s}} \sum_{e=1}^{E} \mathrm{KL}\Big(p_e(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s}) || p_c(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s})\Big), \tag{2.2}$$

where $p_c(\boldsymbol{u}_t, \boldsymbol{x}_t|\boldsymbol{s})$ is the consensus model which can be found by an average or fusion and $\mathrm{KL}(p||q)$ is the KL divergence between probability distribution p and q. Some other techniques consist in reducing the variance assuming a regression problem, which is in general not tractable. Simplifications can result in using Fisher Information and Cramér-Rao inequality for variance reduction techniques such as in [52]. All the aforementioned methods are myopic in the sense that they only care about the information content of single data instance, which can result in selecting outliers or exploring far away in the context space where no generalization is required. Information density method overcomes this problem by choosing instances that have high information content and still representative of the underlying distribution, using a weighted product of uncertainty measure and similarity measure as

$$\boldsymbol{s}^{*}_{ID} = \operatorname*{arg\,max}_{\boldsymbol{s}} \Phi(\boldsymbol{s}) \left(\int_{\tilde{\boldsymbol{s}} \in \mathcal{U}} \Psi(\boldsymbol{s}, \tilde{\boldsymbol{s}}) d\tilde{\boldsymbol{s}} \right)^{\beta}, \tag{2.3}$$

where $\Phi(\boldsymbol{s})$ represents an uncertainty measure (entropy, ensemble, etc.) of context variable $\boldsymbol{s}$, $\Psi(\boldsymbol{s}, \tilde{\boldsymbol{s}})$ represents a similarity measure (Euclidean distance, correlation coefficients, etc.) between the context variable $\boldsymbol{s}$ and all unlabeled potential context variables $\tilde{\boldsymbol{s}}$ from the unlabeled context space $\mathcal{U}$, and β measures the relative weight on the similarity measure over the uncertainty measure [17].

2.2 Optimal Control

Robot behavior primitives learned with state-action abstractions give a probabilistic controller, with no guarantee of stability, unless explicitly constrained to be stable. Pignat et al. [37] propose to fuse their probabilistic non stable controller with a stable one that is attracted to the demonstration. On the contrary, trajectory-level abstractions provide desired states and precisions to a stable optimal controller such as in [53]. Optimal control minimizes a cost function of the state and the actions $c(\cdot)$ to find optimal control commands to achieve desired states given the dynamics model, and can be described as

$$\min_{\boldsymbol{u}_{0,\ldots,T-1}} c(\boldsymbol{x}_{1,\ldots,T}, \boldsymbol{u}_{0,\ldots,T-1}) \quad \text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{2.4}$$

where T is the number of timesteps in the execution. Model Predictive Control (MPC) is one of the most popular optimal controllers [54], because it can handle model uncertainties and adapt to the changes by iteratively replanning at every timestep. However, computational efficiency depends on the nonlinearities contained in the cost function $c(\cdot)$ and the dynamics model $\boldsymbol{f}(\cdot)$. Other intelligent controllers such as variable impedance controller and adaptive controller are used mainly to cope with unexpected disturbances from the environment.

2.2.1 Linear quadratic tracking with least-squares

Linear system evolution

The linear system evolution of the dynamics $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t$ in stacked vector form can be written with the expression $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}$. To find this, we begin by writing

$$
\begin{aligned}
\boldsymbol{x}_1 &= \boldsymbol{A}_0 \boldsymbol{x}_0 + \boldsymbol{B}_0 \boldsymbol{u}_0,\\
\boldsymbol{x}_2 &= \boldsymbol{A}_1 \boldsymbol{x}_1 + \boldsymbol{B}_1 \boldsymbol{u}_1 = \boldsymbol{A}_1 (\boldsymbol{A}_0 \boldsymbol{x}_0 + \boldsymbol{B}_0 \boldsymbol{u}_0) + \boldsymbol{B}_1 \boldsymbol{u}_1,\\
&\vdots\\
\boldsymbol{x}_T &= \prod_{t=0}^{T-1} \boldsymbol{A}_{T-t-1} \boldsymbol{x}_0 + \prod_{t=0}^{T-2} \boldsymbol{A}_{T-t-1} \boldsymbol{B}_0 \boldsymbol{u}_0 + \prod_{t=0}^{T-3} \boldsymbol{A}_{T-t-1} \boldsymbol{B}_1 \boldsymbol{u}_1 + \cdots + \boldsymbol{B}_{T-1} \boldsymbol{u}_{T-1},
\end{aligned}
$$

in a matrix form, we get an expression of the form $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}$, with

$$
\underbrace{\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \vdots \\ \boldsymbol{x}_T \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \boldsymbol{A}_0 \\ \vdots \\ \prod_{t=0}^{T-1} \boldsymbol{A}_{T-t-1} \end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{x}}} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} \boldsymbol{B}_0 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{A}_1 \boldsymbol{B}_0 & \boldsymbol{B}_1 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{t=0}^{T-2} \boldsymbol{A}_{T-t-1} \boldsymbol{B}_0 & \prod_{t=0}^{T-3} \boldsymbol{A}_{T-t-1} \boldsymbol{B}_1 & \cdots & \boldsymbol{B}_{T-1} \end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{u}}} \underbrace{\begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{T-1} \end{bmatrix}}_{\boldsymbol{u}}, \tag{2.5}
$$

by stacking the decision variables $\boldsymbol{x}_{1,\ldots,T}$, $\boldsymbol{u}_{0,\ldots,T-1}$ into $\boldsymbol{x} = \left[\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_T^\top\right]^\top$, $\boldsymbol{u} = \left[\boldsymbol{u}_0^\top, \ldots, \boldsymbol{u}_{T-1}^\top\right]^\top$.

2.2.2 Batch-LQT

The following LQT problem

$$
\begin{aligned}
\min_{\boldsymbol{x}_t, \boldsymbol{u}_t} \quad & \textstyle\sum_{t=0}^{T} (\boldsymbol{x}_t - \boldsymbol{\mu}_{\boldsymbol{x}})^\top \boldsymbol{Q}_t (\boldsymbol{x}_t - \boldsymbol{\mu}_{\boldsymbol{x}}) + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t \\
\text{s.t.} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{A} \boldsymbol{x}_t + \boldsymbol{B} \boldsymbol{u}_t
\end{aligned} \tag{2.6}
$$

can be transformed to its stacked (batch) version using $\boldsymbol{Q}$=blkdiag$(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R}$=blkdiag$(\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_T)$ as

$$
\begin{aligned}
\min_{\boldsymbol{x}, \boldsymbol{u}} \quad & \|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}\|_{\boldsymbol{R}}^2 \\
\text{s.t.} \quad & \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_0 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u},
\end{aligned} \tag{2.7}
$$

which can be solved for $\boldsymbol{u}$ analytically by replacing $\boldsymbol{x}$ in the cost function by the equality constraint and transforming the cost function to be independent of $\boldsymbol{x}$ as $J_{\text{batch-lqt}} = \|\boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_0 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u} - \boldsymbol{x}_d\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u} - \boldsymbol{u}_d\|_{\boldsymbol{R}}^2$. This is a quadratic function of $\boldsymbol{u}$ and can be solved for $\boldsymbol{u}$ in the least-squares sense as in

$$\hat{\boldsymbol{u}} = (\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1}\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}(\boldsymbol{\mu}_{\boldsymbol{x}} - \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0)$$

2.3 Robot Dynamics

The rigid-body dynamics of a robot manipulator with n degrees of freedom is described as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{u} + \boldsymbol{\tau}_{\text{ext}}, \tag{2.8}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are the joint positions, velocities and accelerations, respectively. The inertia matrix $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$, the Coriolis and centrifugal matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ and the gravitational term $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ are the state-dependent parameters of the dynamics model, with state represented by $\boldsymbol{y}{=}[\boldsymbol{q}^{\top}, \dot{\boldsymbol{q}}^{\top}]^{\top}$. The torque-control action $\boldsymbol{u} \in \mathbb{R}^n$ and the external torques $\boldsymbol{\tau}_{\text{ext}} \in \mathbb{R}^n$ acting on the robot form the input to the intrinsic robot dynamics.

Optimization of Controllers Part I

3 Projection-based first-order constrained optimization solver for robotics

Many tasks in robotics can be framed as constrained optimization problems. The inverse kinematics (IK) problem finds a configuration of the robot that corresponds to a desired pose in the task space while satisfying constraints such as joint limits or center-of-mass stability. Motion planning and optimal control determine a trajectory of configurations and/or control commands achieving the task subject to the dynamics and the constraints of the task and the environment over a certain time horizon. Model predictive control (MPC) recasts the optimal control problems with shorter horizons to solve simpler constrained optimization problems in real-time. In this work, we present a projection-based first-order optimization method that can be implemented and used for all these aforementioned problems.

There are many commercially available second-order solvers to address general constrained optimization problems such as SNOPT [5], SLSQP [6], LANCELOT [7] and IPOPT [8]. However, in robotics, the literature on optimization is still focusing on developing solvers that effectively solve each robotic problem separately. For example, in the motion planning literature, one can find many constrained variants of differential dynamic programming (DDP) [9] or iterative linear quadratic regulator (iLQR) [10], TrajOpt [12], CHOMP [13] and ALTRO [11]. Furthermore, some of these solvers are not easily adaptable and/or difficult to implement, which hinders benchmarking and potential improvements. As powerful as these solvers are, their applications for finding real-time feedback mechanisms such as closed-loop inverse kinematics and MPC requires tuning and adaptations of the solver. In this chapter, we address these challenges by proposing a very simple, yet powerful solver that can be easily implemented without having large memory requirements.

The constraints in many of these problems are described as geometric set primitives or their combinations (see Table 3.1). Examples include joint angle or velocity limits or center-of-mass stability as bounded domain sets, avoiding/reaching geometric shapes such as spheres and convex polytopes as hyperplane and quadric sets, friction cone constraints as second-order cone sets. These constraints have in common that they can be formulated as projections rather than constraints. This chapter argues that exploiting the projection capability of these sets instead of treating them as generic constraints in the solvers significantly improves the performance.

Projected gradient descent is the simplest algorithm that takes into account these projections. Its idea is to project the gradient to have a next iterate inside the constraint set. In the optimization literature, a first-order projection-based solver called spectral projected gradient descent (SPG) emerges as an alternative [55]. SPG has been studied and applied to many fields because of its great practical performance even compared to second-order constrained optimization solvers [14]. Its extension to additional arbitrary constraints has been proposed as within augmented Lagrangian methods [56, 57, 58]. However, as the application of this idea to popular second-order methods in robotics is not trivial [59], the usefulness of projections in the field has been overlooked.

In this chapter, we integrate the recent work in [58] into robotics optimization problems ranging from IK to MPC by providing the most common Euclidean projections with an additional rectangular projection. We propose an extension with multiple projections and additional nonlinear constraints. In particular, we provide an efficient direct-shooting optimal control formulation of this solver to address motion planning and MPC problems.

3.1 Related work

Euclidean projections and the analytical expressions to many projections can be found in the thesis [60]. It also gives a general theory on how to project onto a level set of an arbitrary function using KKT conditions. Extensive studies and theoretical background on the projections and their properties can be found in [61]. In [62], the authors propose an efficient algorithm for the projection onto arbitrary convex constraint sets and show that exploiting projections in the optimization significantly increases the performance. [63] discusses and benchmarks algorithms for finding the projection onto the intersection of convex sets. One of the main algorithms for this is Dykstra's alternating projection algorithm [64].

The simplest algorithm that exploits projections is the projected gradient descent. Spectral projected gradient descent improves over this by exploiting the curvature information via its spectral stepsizes. A detailed review on spectral projected gradient methods is given in [57].

In [65], the authors propose to use a projected gradient descent algorithm to solve the subproblems of sequential quadratic programming (SQP). They show that their method can solve MPC of an inverted pendulum faster than SNOPT. Our work is closest to theirs with the differences that we use SPG instead of a vanilla projected gradient descent

Table 3.1: Projections onto bounded domain, affine hyperplane, quadric and second-order cone

	Bounds	Affine hyperplane	Quadric	Second-order cone
$\mathcal{C}$	$l{\le}x{\le}u$	$l{\le}\boldsymbol{a}^\top\boldsymbol{x}{\le}u$	$l{\le}\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x}{\le}u$	$\lVert\boldsymbol{x}\rVert{\le}t$
$\Pi_{\mathcal{C}}$	$\begin{cases} x & \text{if } l{\le}x{\le}u \\ u & \text{if } x{>}u \\ l & \text{if } x{<}l \end{cases}$	$\begin{cases} \boldsymbol{x} & \text{if } l{\le}\boldsymbol{a}^\top\boldsymbol{x}{\le}u \\ \boldsymbol{x}-\frac{\boldsymbol{a}(\boldsymbol{a}^\top\boldsymbol{x}-u)}{\lVert\boldsymbol{a}\rVert_2^2} & \text{if } \boldsymbol{a}^\top\boldsymbol{x}{>}u \\ \boldsymbol{x}-\frac{\boldsymbol{a}(\boldsymbol{a}^\top\boldsymbol{x}-l)}{\lVert\boldsymbol{a}\rVert_2^2} & \text{if } \boldsymbol{a}^\top\boldsymbol{x}{<}l \end{cases}$	$\begin{cases} \boldsymbol{x} & \text{if } l{\le}\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x}{\le}u \\ \frac{\boldsymbol{x}\sqrt{2u}}{\lVert\boldsymbol{x}\rVert} & \text{if } \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x}{>}u \\ \frac{\boldsymbol{x}\sqrt{2l}}{\lVert\boldsymbol{x}\rVert} & \text{if } l{>}\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x} \end{cases}$	$\begin{cases} (\boldsymbol{x},t), & \text{if } \lVert\boldsymbol{x}\rVert{\le}t \\ (\boldsymbol{0},0), & \text{if } \lVert\boldsymbol{x}\rVert{\le}-t \\ \frac{\lVert\boldsymbol{x}\rVert+t}{2}(\frac{\boldsymbol{x}}{\lVert\boldsymbol{x}\rVert},1) & \text{otherwise} \end{cases}$

to solve the subprolems of augmented Lagrangian instead of SQP. Instead, we propose a direct way of handling multiple projections and inequality constraints, which is not trivial in [65].

In [66], authors propose a projection of the update direction of the control input onto the nullspace of the linearized constraints in iLQR. This approach can only handle simple equality constraints (for example, velocity-level constraints of second-order systems) and cannot treat position-level constraints for such systems, which is a very common and practical class of constraints in real world applications.

3.2 Background

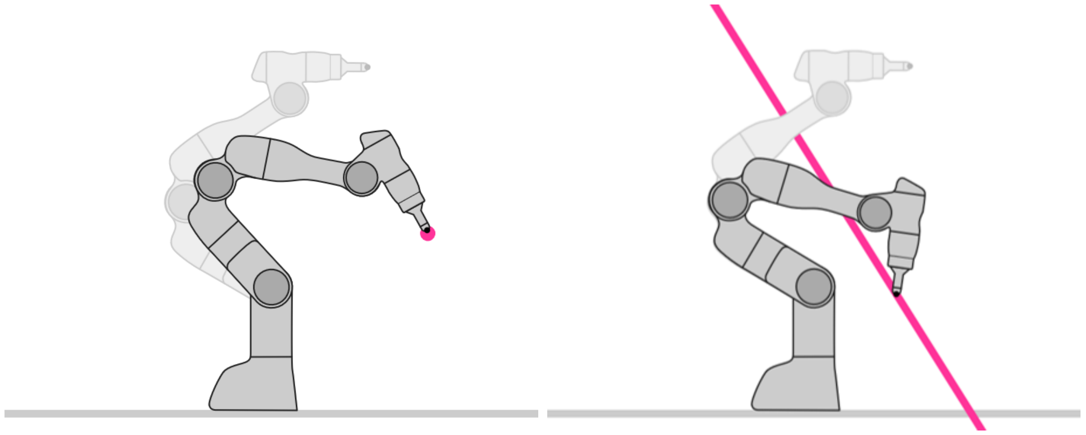
In this section, we give the background on the projections motivating their use in standard robotic tasks such as hierarchical inverse kinematics and obstacle avoidance.

3.2.1 Euclidean projections onto sets

The solution $\boldsymbol{x}^*$ to the following constrained optimization problem

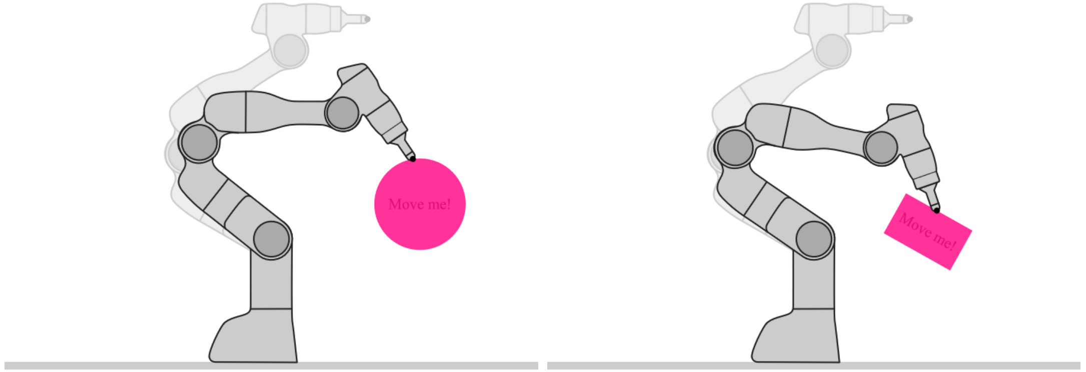
$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \lVert\boldsymbol{x}-\boldsymbol{x}_0\rVert_2^2 \\ \text{s.t.} \quad & \boldsymbol{x}\in\mathcal{C} \end{aligned} \tag{3.1}$$

is called an Euclidean projection of the point $\boldsymbol{x}_0$ onto the set $\mathcal{C}$ and is denoted as $\boldsymbol{x}^*{=}\Pi_{\mathcal{C}}(\boldsymbol{x}_0)$. This operation determines the point $\boldsymbol{x}\in\mathcal{C}$ that is closest to $\boldsymbol{x}_0$ in Euclidean sense. For many sets $\mathcal{C}$, $\Pi_{\mathcal{C}}(\cdot)$ admits analytical expressions that are given in Table 3.1. Even though, usually these sets are convex (e.g. bounded domains), some nonconvex sets also admit analytical solution(s) that are easy to compute (e.g. being outside of a sphere). Note that many of these sets are frequently used in robotics, from joint/torque limits and avoiding spherical/square obstacles to satisfying virtual fixtures defined in the task space of the robot.



(a) Reaching a point (standard IK problem): $\mathcal{C}_{\boldsymbol{p}} = \{\boldsymbol{p} \mid \boldsymbol{p} = \boldsymbol{p}_d\}$

(b) Reaching under/above/on a plane (in the halfspace): $\mathcal{C}_{\boldsymbol{p}} = \{\boldsymbol{p} \mid \boldsymbol{a}^\top \boldsymbol{p} \le \boldsymbol{b}\}$



(c) Reaching inside/outside/on a circle: $\mathcal{C}_{\boldsymbol{p}} = \{\boldsymbol{p} \mid \|\boldsymbol{p} - \boldsymbol{p}_d\|_2^2 \le r^2\}$

(d) Reaching inside/outside/on a rectangle: $\mathcal{C}_{\boldsymbol{p}} = \{\boldsymbol{p} \mid \|\boldsymbol{A}(\boldsymbol{p} - \boldsymbol{p}_d)\|_\infty \le L/2\}$

Figure 3.1: Projection view of inverse kinematics problem. These problems can be tested online with closed-loop controllers on the provided website[1]created as extensions of the toolbox Robotics Codes from Scratch[2].

3.2.2 Projection view of inverse kinematics

The inverse kinematics (IK) problem in robotics corresponds to finding a joint configuration $\boldsymbol{q}^*$ of the robot that corresponds to a given desired end-effector pose $\boldsymbol{p}_d$. Iterative procedures are developed to robustly solve this problem considering singularities at the Jacobian level. The success and the convergence speed of these algorithms depend on the initialization of the problem, which is often selected as the current joint configuration of the robot $\boldsymbol{q}_0$. With this view in mind, we can express IK as a projection problem of the initial joint angles $\boldsymbol{q}_0$ onto a set $\mathcal{C}_{\boldsymbol{q}}$ and of the initial end-effector position $\boldsymbol{p}_0$ onto a set $\mathcal{C}_{\boldsymbol{p}}$. These two sets are assumed to be nonempty and closed sets that admit tractable and efficient projections. A common example for $\mathcal{C}_{\boldsymbol{q}}$ is the box constraints for the joint limits. Figure 3.1 shows examples for the set $\mathcal{C}_{\boldsymbol{p}}$ with Figure 3.1a showing an equality constraint to a desired point, Figure 3.1b shows an affine hyperplane constraint for virtually limiting the robot to be under/on a plane, Figure 3.1c and Figure 3.1d show quadric constraints for the end-effector to stay inside/outside or on the boundary of a circle/square. In this work, we exploit these easy projections in a first-order optimization solver with the claim of finding solutions faster than standard constrained optimization problems.

3.3 Augmented Lagrangian Spectral Projected Gradient Descent for Robotics

This section gives the spectral projected gradient descent (SPG) algorithm along with the nonmonotone line search procedure. These algorithms are easy to implement without big memory requirements and yet result in powerful solvers. Next, we give the augmented Lagrangian spectral projected gradient descent (ALSPG) algorithm with extensions to general inequality constraints and multiple projections.

3.3.1 Spectral projected gradient descent

Spectral projected gradient descent (SPG) is an improved version of a vanilla projected gradient descent using spectral stepsizes. Its excellent numerical results even in comparison to second-order methods have been a point of attraction in the optimization literature [14]. SPG tackles constrained optimization problems in the form of

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{C}, \end{aligned} \tag{3.2}$$

by constructing a local quadratic model of the objective function

$$f(\boldsymbol{x}) \approxeq f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^\top (\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2\gamma_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 = \frac{1}{2\gamma_k}\|\boldsymbol{x} - (\boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k))\|_2^2 + \text{const.}$$

and by minimizing it subject to the constraints as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \tfrac{1}{2\gamma_k}\|\boldsymbol{x} - (\boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k))\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{C}, \end{aligned} \tag{3.3}$$

whose solution is an Euclidean projection as described in Section 3.2.1 and given by $\Pi_{\mathcal{C}}(\boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k))$. The local search direction $\boldsymbol{d}_k$ for SPG is then given by

$$\boldsymbol{d}_k = \Pi_{\mathcal{C}}(\boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k)) - \boldsymbol{x}_k, \tag{3.4}$$

---

[1] https://hgirgin.github.io/IKSPG.html

[2] https://robotics-codes-from-scratch.github.io/

which is used in a nonmonotone line search (Algorithm 1) with $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$, to find α_k satisfying $f(\boldsymbol{x}_{k+1}) \leq f_{\max} + \alpha_k \gamma_k \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{d}_k$, where $f_{\max} = \max\{f(\boldsymbol{x}_{k-j}) \mid 0 \leq j \leq \min\{k, M-1\}\}$. Nonmonotone line search allows for increasing objective values for some iterations M preventing getting stuck at bad local minima.

The choice of γ_k affects the convergence properties significantly since it introduces curvature information to the solver. Note that when choosing $\gamma_k = 1$, SPG is equivalent to the widely known projected gradient descent. SPG uses spectral stepsizes obtained by a least-square approximation of the Hessian matrix by $\gamma_k \boldsymbol{I}$. These spectral stepsizes are computed by

$$\gamma_k^{(1)} = \frac{\boldsymbol{s}_k^\top \boldsymbol{s}_k}{\boldsymbol{s}_k^\top \boldsymbol{y}_k} \quad \text{and} \quad \gamma_k^{(2)} = \frac{\boldsymbol{s}_k^\top \boldsymbol{y}_k}{\boldsymbol{y}_k^\top \boldsymbol{y}_k}, \tag{3.5}$$

where $\boldsymbol{s}_k = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$ and $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1})$ [14]. In the case of quadratic objective function in the form of $\boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x}$, these two values correspond to the maximum and minimum eigenvalues of the matrix $\boldsymbol{Q}$. Recent developments in SPG have shown that an alternating use of these spectral stepsizes lead to a better performances. The initial spectral stepsize can be computed by setting $\bar{\boldsymbol{x}}_0 = \boldsymbol{x}_0 - \gamma_{\text{small}} \nabla f(\boldsymbol{x}_0)$, and computing $\bar{\boldsymbol{s}}_0 = \bar{\boldsymbol{x}}_0 - \boldsymbol{x}_0$ and $\bar{\boldsymbol{y}}_0 = \nabla f(\bar{\boldsymbol{x}}_0) - \nabla f(\boldsymbol{x}_0)$. Note that this heuristic operation costs one more gradient computation. The final algorithm is then given by Algorithm 2.

Algorithm 1: Non-monotone line search

```
Set $\beta = 10^{-4}$, $\alpha = 1$, $M = 10$, $c = \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{d}_k$,
$f_{\max} = \max\{f(\boldsymbol{x}_{k-j}) | 0 \leq j \leq \min\{k, M-1\}\}$
while $f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) > f_{max} + \alpha \beta c$ do
    $\bar{\alpha} = -0.5\alpha^2 c \Big( f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) - f(\boldsymbol{x}_k) - \alpha c \Big)^{-1}$
    if $0.1 \leq \bar{\alpha} \leq 0.9$ then
        $\alpha = \bar{\alpha}$
    else
        $\alpha = \alpha/2$
    end
end
```

3.3.2 Augmented Lagrangian spectral projected gradient descent (AL-SPG)

SPG algorithm has been shown to be a powerful competition to second-order solvers in many ways. Each iteration can be significantly cheaper than a second-order method if a computationally efficient projection is used and provides better directions than other first-order methods. However, SPG alone is usually not sufficient to solve problems in robotics with complicated nonlinear constraints. [58] provides an augmented Lagrangian

Algorithm 2: Spectral Projected Gradient Descent

```
Initialize $\boldsymbol{x}_k$, $\gamma_k$ $\epsilon$=$10^{-5}$, $k$=0;
while $\|\Pi_{\mathcal{C}}(\boldsymbol{x}_k - \nabla f(\boldsymbol{x}_k)) - \boldsymbol{x}_k\|_\infty > \epsilon$ do
    Find a search direction by $\boldsymbol{d}_k = \Pi_{\mathcal{C}}(\boldsymbol{x}_k - \gamma_k \nabla f(\boldsymbol{x}_k)) - \boldsymbol{x}_k$
    Do non-monotone line search using Algorithm 1 to find $\boldsymbol{x}_{k+1}$
    Update the spectral stepsize
    $\boldsymbol{s}_{k+1} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_{k+1} = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$
    $\gamma^{(1)} = \frac{\boldsymbol{s}_{k+1}^\top \boldsymbol{s}_{k+1}}{\boldsymbol{s}_{k+1}^\top \boldsymbol{y}_{k+1}}$ and $\gamma^{(2)} = \frac{\boldsymbol{s}_{k+1}^\top \boldsymbol{y}_{k+1}}{\boldsymbol{y}_{k+1}^\top \boldsymbol{y}_{k+1}}$
    if $\gamma^{(1)} < 2\gamma^{(2)}$ then
        $\gamma_{k+1} = \gamma^{(2)}$
    else
        $\gamma_{k+1} = \gamma^{(1)} - \frac{1}{2}\gamma^{(2)}$
    end
    $k = k + 1$
end
```

framework to solve problems with constraints $\boldsymbol{g}(\boldsymbol{x}) \in \mathcal{C}$ and $\boldsymbol{x} \in \mathcal{D}$, where $\boldsymbol{g}(\cdot)$ is a convex function, $\mathcal{C}$ is a convex set, and $\mathcal{D}$ is closed nonempty set, both equipped with easy projections.

In this section, we build on the work in [58] with the extension of multiple projections and additional general equality and inequality constraints. The general optimization problem that we are tackling here is

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathcal{D}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}_i(\boldsymbol{x}) \in \mathcal{C}_i, \quad \forall i \in \{1, \dots, p\} \end{aligned} \tag{3.6}$$

where $\boldsymbol{g}_i(\cdot)$ are assumed to be arbitrary nonlinear functions. Note that even though the convergence results in [58] apply to the case when these are convex functions and convex sets, we found in practice that the algorithm is powerful enough to extend to more general cases. For simplicity, we redefine the additional equality constraints as an additional set to be projected onto with $\mathcal{C}_y = \{\boldsymbol{y} \mid \boldsymbol{y} = \boldsymbol{0}\}$ with $\Pi_{\mathcal{C}_g}(\boldsymbol{h}(\cdot)) = \boldsymbol{0}$. Also, we transform inequality constraints to equality constraints using the proposed method in the following Section 3.3.3.

We use the following augmented Lagrangian function

$$\mathcal{L}(\boldsymbol{x}, \{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p}) = f(\boldsymbol{x}) + \sum_{i=1}^{p} \frac{\rho^{\mathcal{C}_i}}{2} \left\| \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}} - \Pi_{\mathcal{C}_i}\Big(\boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}}\Big) \right\|_2^2 \tag{3.7}$$

whose derivative wrt $\boldsymbol{x}$ is given by

$$\nabla \mathcal{L}(\boldsymbol{x}, \{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p}) = \nabla f(\boldsymbol{x}) + \sum_{i=1}^{p} \frac{\rho^{\mathcal{C}_i}}{2} \nabla \boldsymbol{g}_i^{\top}(\boldsymbol{x}) \Big(\boldsymbol{g}_i(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}} - \Pi_{\mathcal{C}_i}\Big(\boldsymbol{g}_i(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}} \Big) \Big),$$

using the property of convex Euclidean projections derivative [61]

$$\nabla \|\boldsymbol{g}(\boldsymbol{x}) - \Pi(\boldsymbol{g}(\boldsymbol{x}))\|_2^2 = \nabla \boldsymbol{g}(\boldsymbol{x})^{\top} \Big(\boldsymbol{g}(\boldsymbol{x}) - \Pi(\boldsymbol{g}(\boldsymbol{x})) \Big). \tag{3.8}$$

This way, we obtain a formulation which does not need the gradient of the projection function $\Pi_{\mathcal{C}_i}(\cdot)$. One iteration of ALSPG optimizes the subproblem $\mathsf{arg\,min}_{\boldsymbol{x} \in \mathcal{D}} \mathcal{L}(\boldsymbol{x}, \{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p})$ given $\{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p}$, and then updates these according to the next iterate. Defining the auxiliary function $V(\boldsymbol{x}, \boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}) = \Big\| \boldsymbol{g}(\boldsymbol{x}) - \Pi_{\mathcal{C}_i}\Big(\boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}} \Big) \Big\|$, the algorithm is summarized in Algorithm 3. Note that one can define and tune many heuristics around augmented Lagrangian methods with possible extensions to primal-dual methods and here we give only one possible way of implementing ALSPG.

Algorithm 3: ALSPG

```
Set $\boldsymbol{\lambda}_0^{\mathcal{C}_i}=\mathbf{0}$, $\rho_0^{\mathcal{C}_i}=0.1$, $k=0$, $\epsilon > 0$
while $\|\Delta \mathcal{L}(\boldsymbol{x}, \{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p})\| > \epsilon$ do
    $\boldsymbol{x}_{k+1} = \mathsf{arg\,min}_{\boldsymbol{x} \in \mathcal{D}} \mathcal{L}(\boldsymbol{x}, \{\boldsymbol{\lambda}^{\mathcal{C}_i}, \rho^{\mathcal{C}_i}\}_{i=1}^{p})$ with SPG in Algorithm 2
    foreach $\mathcal{C}_i$ do
        $\lambda_{k+1}^{\mathcal{C}_i} = \rho^{\mathcal{C}_i}\Big(\boldsymbol{g}_i(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}} - \Pi_{\mathcal{C}_i}\Big(\boldsymbol{g}_i(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}^{\mathcal{C}_i}}{\rho^{\mathcal{C}_i}}\Big)\Big)$
        if $V(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}^{\mathcal{C}_i}, \rho_k^{\mathcal{C}_i}) \leq V(\boldsymbol{x}_k, \boldsymbol{\lambda}_k^{\mathcal{C}_i}, \rho_k^{\mathcal{C}_i})$ then
            $\rho_{k+1}^{\mathcal{C}_i}=\rho_k^{\mathcal{C}_i}$
        else
            $\rho_{k+1}^{\mathcal{C}_i}=10\rho_k^{\mathcal{C}_i}$
        end
    end
end
```

3.3.3 Handling of inequality constraints

In robotics, one frequent and intuitive way of incorporating the constraints into the optimization problem is to use soft constraints and tune the weights until a satisfactory result is obtained. However, this approach breaks the hierarchy of the task without any real guarantee of constraint satisfaction. Soft constraints are obtained by transforming the hard constraint function into a positive cost function using auxiliary functions such as the barrier function. In this section, we propose to exploit such soft constraint functions as hard constraints to reduce each inequality constraint to an equality constraint, eliminating the need of using slack variables. Note that this procedure is in line with the construction of a standard augmented Lagrangian for inequality constraints.

Let $g_i(\boldsymbol{x}) \leq 0$ be the i^{th} inequality constraint with $i{=}1,\dots,M$ and $g(\cdot) : \mathbb{R}^n \to \mathbb{R}$. We define $h_i(\boldsymbol{x}) = \max(0, g_i(\boldsymbol{x})$, where $h(\cdot) : \mathbb{R}^n \to \mathbb{R}^+$. Then, the statement $g_i(\boldsymbol{x})) \leq 0$ is equivalent to $h_i(\boldsymbol{x}) = 0$. Moreover, we can generalize this statement to obtain one single equality constraint from any number of inequality constraints in order to increase the computational speed. This generalization is given by the theorem below.

Theorem 1. *The statement* $g_i(\boldsymbol{x}) \leq 0, \forall i{=}1,\dots,M$ *is equivalent to* $h(\boldsymbol{x}) = \sum_{i=1}^{M} h_i(\boldsymbol{x}) = 0$, *where* $h_i(\boldsymbol{x}) = \max(0, g_i(\boldsymbol{x}))$.

Proof.

1. If $g_i(\boldsymbol{x}) \leq 0, \forall i{=}1,\dots,M$, then it is by definition that $\sum_{i=1}^{M} h_i(\boldsymbol{x}) = 0$.
2. Assume $\sum_{i=1}^{M} h_i(\boldsymbol{x}) = 0$ and $\exists j \quad \text{s.t.} \quad g_j(\boldsymbol{x}) > 0, \quad \forall j{=}1,\dots,N < M$. Then, $\sum_{i=1}^{M} h_i(\boldsymbol{x}) = \sum_{j=1}^{N} h_j(\boldsymbol{x}) = \sum_{j=1}^{N} g_j(\boldsymbol{x}) > 0$, which contradicts the assumption.

□

Although it seems to simplify the problem a lot in terms of dimensions, using Theorem 1 to reduce compactly all inequality constraints into one single constraint would lose some contribution about the gradients from each constraint in one iteration of any solver. In practice, this presents itself as a trade-off between the number of iterations and the computational complexity of each iteration to solve the optimization problem.

3.3.4 Optimal Control with ALSPG

We consider the following generic constrained optimization problem

$$\begin{aligned} \min_{x \in \mathcal{C}_{\boldsymbol{x}}, \boldsymbol{u} \in \mathcal{C}_{\boldsymbol{u}}} \quad & c(\boldsymbol{x}, \boldsymbol{u}) \\ \text{s.t.} \quad & \boldsymbol{x} = \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}), \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{0}, \end{aligned} \tag{3.9}$$

where $\boldsymbol{x} = \left[\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \dots, \boldsymbol{x}_t^\top, \dots, \boldsymbol{x}_T^\top\right]^\top$, $\boldsymbol{u} = \left[\boldsymbol{u}_0^\top, \boldsymbol{u}_1^\top, \dots, \boldsymbol{u}_t^\top, \dots, \boldsymbol{u}_{T-1}^\top\right]^\top$ and the function $\boldsymbol{F}(\cdot, \cdot)$ correspond to the forward rollout of the states using a dynamics model $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, namely, $\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}) = \left[\boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{u}_0)^\top, \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{u}_1)^\top, \dots, \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)^\top, \dots, \boldsymbol{f}(\boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1})^\top\right]^\top$. We use a direct shooting approach and transform Equation (3.9) into a problem in $\boldsymbol{u}$ only by

$$\begin{aligned} \min_{\boldsymbol{u} \in \mathcal{C}_{\boldsymbol{u}}} \quad & c(\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}), \boldsymbol{u}) \\ \text{s.t.} \quad & \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}) \in \mathcal{C}_{\boldsymbol{x}}, \\ & \boldsymbol{h}(\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}), \boldsymbol{u}) = \boldsymbol{0}, \end{aligned} \tag{3.10}$$

which is exactly in the form of Equation (3.6), if $\boldsymbol{g}_1(\boldsymbol{u}) = \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})$ and $\boldsymbol{g}_2(\boldsymbol{u}) = \boldsymbol{h}(\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}), \boldsymbol{u})$. The unconstrained version of this problem can be solved with least-square approaches. However, assuming $\boldsymbol{x}_t \in \mathbb{R}^m$, $\boldsymbol{u}_t \in \mathbb{R}^n$, this requires an inversion of a matrix of size $Tn \times Tn$, whereas here we only work with the gradients of the objective function and the functions $\boldsymbol{g}_i(\cdot)$. The component that requires a special attention is $\nabla \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})$ and in particular, its tranpose product with a vector. It turns out that this product can be efficiently computed with a recursive formula (as also described in [65]), resulting in fast SPG iterations. Denoting $\boldsymbol{A}_t = \nabla_{\boldsymbol{x}_t} \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and $\boldsymbol{B}_t = \nabla_{\boldsymbol{u}_t} \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, and $\nabla_{\boldsymbol{u}} \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})^\top \boldsymbol{y} = \boldsymbol{z}$ with $\boldsymbol{y} = \Big[\boldsymbol{y}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_t, \dots, \boldsymbol{y}_{T-1}\Big]$, $\boldsymbol{z} = \Big[\boldsymbol{z}_0, \boldsymbol{z}_1, \dots, \boldsymbol{z}_t, \dots, \boldsymbol{z}_{T-1}\Big]$, one can show that

$$\nabla_{\boldsymbol{u}} \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})^\top \boldsymbol{y} = \begin{bmatrix} \boldsymbol{B}_0^\top & \boldsymbol{B}_0^\top \boldsymbol{A}_1^\top & \boldsymbol{B}_0^\top \boldsymbol{A}_1^\top \boldsymbol{A}_2^\top & \dots & \boldsymbol{B}_0^\top \prod_{t=1}^{T-1} \boldsymbol{A}_t^\top \\ \boldsymbol{0} & \boldsymbol{B}_1^\top & \boldsymbol{B}_1^\top \boldsymbol{A}_2^\top & \dots & \boldsymbol{B}_1^\top \prod_{t=2}^{T-1} \boldsymbol{A}_t^\top \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{B}_{T-1}^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_{T-1} \end{bmatrix} \tag{3.11}$$

$$= \begin{bmatrix} \boldsymbol{B}_0^\top (\boldsymbol{y}_0 + \boldsymbol{A}_1^\top \boldsymbol{y}_1 + \boldsymbol{A}_1^\top \boldsymbol{A}_2^\top \boldsymbol{y}_2 + \prod_{t=1}^{T-1} \boldsymbol{A}_t^\top \boldsymbol{y}_{T-1}) \\ \boldsymbol{B}_1^\top (\boldsymbol{y}_1 + \boldsymbol{A}_2^\top \boldsymbol{y}_2 + \boldsymbol{A}_2^\top \boldsymbol{A}_3^\top \boldsymbol{y}_3 + \prod_{t=2}^{T-1} \boldsymbol{A}_t^\top \boldsymbol{y}_{T-1}) \\ \vdots \\ \boldsymbol{B}_{T-2}^\top (\boldsymbol{y}_{T-2} + \boldsymbol{A}_{T-1}^\top \boldsymbol{y}_{T-1}) \\ \boldsymbol{B}_{T-1}^\top \boldsymbol{y}_{T-1} \end{bmatrix} \tag{3.12}$$

where the terms in parantheses can be computed recursively backward by $\bar{\boldsymbol{z}}_{t+1} = (\boldsymbol{y}_{t+1} + \boldsymbol{A}_t^\top \bar{\boldsymbol{z}}_t)$, $\boldsymbol{z}_t = \boldsymbol{B}_{t-1}^\top \bar{\boldsymbol{z}}_t$ and $\bar{\boldsymbol{z}}_{T-1} = \boldsymbol{y}_{T-1}$, without having to construct the big matrix $\nabla_{\boldsymbol{u}} \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})^\top$.

Note that when there are no constraints on the state and $\boldsymbol{h}(\cdot) = 0$, then Equation (3.10) can be solved directly with SPG algorithm. We believe that SPG can be used to solve problems with higher horizons even faster than iLQR. For a breakdown of computational times compared to the number of timesteps for a reaching planning tasks without constraints for 7DoF manipulator, see Figure 3.2. Here we plotted the average convergence time in (s) for both algorithms with 5 different end positions in task space and horizons of 100, 1000, 2000, 3000 and 5000 timesteps. iLQR is implemented with dynamic programming, and SPG as explained in the previous section, both in Python.

3.4 Convex polytope projections and linear transformations

Often, Euclidean projection problems are composed of a linear transformation of the constraint set onto which the projection is easy, hindering the analytical projection property of this set. ALSPG can be used directly to solve for this kind of problems in a very efficient way, still exploiting the projection capability of the base constraint set. For example, consider a unit second-order cone set as $\mathcal{C}_{\text{SOC}} = \{(\boldsymbol{z}, t) \mid \|\boldsymbol{z}\|_2 \leq t\}$ and a generic second-order cone (SOC) constraint as $\mathcal{C}{=}\{\boldsymbol{x} \mid \|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\|_2 \leq \boldsymbol{c}^\top \boldsymbol{x} + d\}$. This

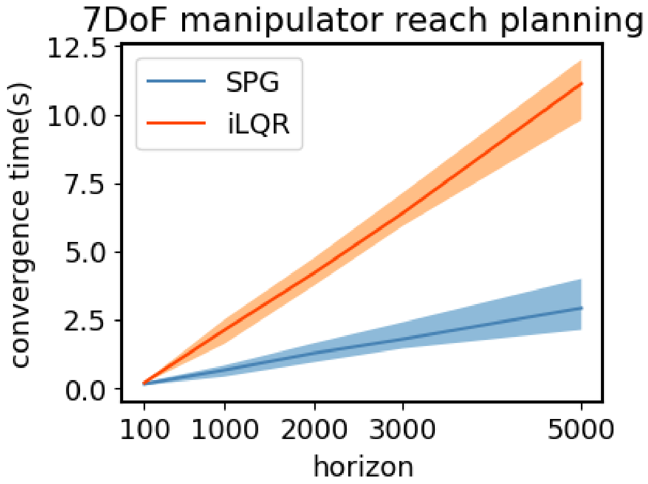


Figure 3.2: Comparison of iLQR and SPGOC in terms of convergence time evolution vs the number of timesteps or horizon.

can be transformed to a unit second-order cone set by taking $\boldsymbol{g}(\boldsymbol{x})=\begin{bmatrix}\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b} & \boldsymbol{c}^\top\boldsymbol{x}+d\end{bmatrix}$ and therefore $\mathcal{C}=\{\boldsymbol{x} \mid \boldsymbol{g}(\boldsymbol{x}) \in \mathcal{C}_{\text{SOC}}\}$. Then the optimization problem of projection onto a generic second-order cone, namely, $\arg\min_{\boldsymbol{x}} \|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}\|_2 \leq \boldsymbol{c}^\top\boldsymbol{x}+d$ can be rewritten as $\arg\min_{\boldsymbol{x}} \|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2 \quad \text{s.t.} \quad \boldsymbol{g}(\boldsymbol{x}) \in \mathcal{C}_{\text{SOC}}$ and can be solved efficiently using ALSPG algorithm with only unit second-order cone projections, without requiring explicit derivatives of the cone constraints.

In the case of convex polytope projections, we can even find some conditions when the linear transformation does not break the analytical projections. Especially for rectangular projections, which are a special case of convex polytope projections, we can find special conditions such that we can still find analytical expressions even if we rotate and scale the rectangles. In the next section, we give the development and insights of convex polytope projections as these are one of the most commonly encountered constraint types in robotics problems such obstacle avoidance. We then explain what kind of linear transformations can be applied to projections to preserve analytical projection capability.

3.4.1 Convex polytope projections

A convex polytope of n sides can be described by n lines with slopes $\boldsymbol{a}_i$ and intercepts b_i. The inside region of this polytope (e.g. for reaching) is given by *and* constraints $\mathcal{C}^{\text{in}}_{\text{polytope}} = \{\boldsymbol{x} | \bigwedge_{i=0}^{n} \boldsymbol{a}_i^\top \boldsymbol{x} \leq u_i\}$ while the outside region (e.g. for obstacle avoidance) is given by its negative statement with *or* constraints $\mathcal{C}^{\text{out}}_{\text{polytope}} = \{\boldsymbol{x} | \bigvee_{i=0}^{n} \boldsymbol{a}_i^\top \boldsymbol{x} > l_i\}$. The projection onto $\mathcal{C}^{\text{in}}_{\text{polytope}}$ can be described as a summation of n hyperplane projections in ALSPG. Even though constraints for the set $\mathcal{C}^{\text{out}}_{\text{polytope}}$ can not be easily described in general optimization solvers, we can show that the projection of a point $\boldsymbol{x}_0$ onto this set requires finding the closest hyperplane i to $\boldsymbol{x}_0$, then the projection outside the hyperplane with index i, namely $\Pi^i_{\mathcal{C}^{\text{out}}_{\text{polytope}}}(\boldsymbol{x}_0)$. The minimum value of the objective function of the projection becomes $\|\Pi^i_{\mathcal{C}^{\text{out}}_{\text{polytope}}}(\boldsymbol{x}_0) - \boldsymbol{x}_0\|$ which is equal to the distance of $\boldsymbol{x}_0$ to the hyperplane i (one can check this by inserting the corresponding values from Table 3.1). This observation makes significant simplifications for the solvers that can take projections into account.

In this section, we give the simplifications of this idea for often-encountered rectangular regions. The constraint of being inside a square region, also called a box constraint, can be described by infinity norms as the set $\mathcal{C}^{\text{in}}_{\text{rect}}=\{\boldsymbol{x}\,|\,\|\boldsymbol{x}\|_\infty \leq u\}$ represents the inside region of a square of width u centered at the origin. This is basically a compact description of 4 lines (in 2D) describing the square, i.e., $x \leq u$, $-u \leq x$, $y \leq u$, $-u \leq y$. This observation allows us to write down $\mathcal{C}^{\text{out}}_{\text{rect}}=\{x \mid l \leq \|\boldsymbol{x}\|_\infty\}$ which represents the outside region of a square of width l centered at the origin. $\mathcal{C}^{\text{in}}_{\text{rect}}$ is a simple clipping operation for $\boldsymbol{x}_0$ as described in Table 3.1. However, $\mathcal{C}^{\text{out}}_{\text{rect}}$ requires setting up the optimization problem for the Euclidean projection and checking the KKT conditions. For conciseness, we give here only the resulting projection. Denoting k the index where $k=\operatorname{arg\,max}_i |\boldsymbol{x}_{0,i}|$, the projection onto $\mathcal{C}^{\text{out}}_{\text{rect}}$ is then given by

$$\Pi_{\mathcal{C}}(\boldsymbol{x}_0)_j = \begin{cases} \boldsymbol{x}_{0,j} & \text{if } \boldsymbol{x}_{0,k} \leq l \\ l \operatorname{sign} \boldsymbol{x}_{0,k} & \text{otherwise} \end{cases} \tag{3.13}$$

3.4.2 Linear transformation of projections

Having stated projections for some basic geometric primitives, one may need to apply rotation and translation operations to such shapes to exploit more complex ones. One such example is the transformation of square projections onto rotated and translated square regions. Considering a convex set $\mathcal{C} = \{\boldsymbol{x}|f(\boldsymbol{x}) \leq t\}$, one can show that the projection onto $\mathcal{C}' = \{\boldsymbol{x}|f(\boldsymbol{A}(\boldsymbol{x}-\boldsymbol{x}_c)) \leq t\}$ is given by $\Pi_{\mathcal{C}'}(\boldsymbol{x}_0) = \boldsymbol{A}^{-1}\Pi_{\mathcal{C}}(\boldsymbol{A}(\boldsymbol{x}_0-\boldsymbol{x}_c))+\boldsymbol{x}_c$, where $\boldsymbol{A}$ is an orthogonal matrix. For creating rectangular regions, one needs to scale each dimension of the variable, i.e. multiplying by a diagonal matrix. Even though this does not generalize to all cases, for the rectangular regions, one can show that $\boldsymbol{A}$ can be in the form of a multiplication of an orthogonal matrix and a diagonal matrix. For example, while a square of length L can be described by the set $\mathcal{C} = \{\boldsymbol{x}|\|\boldsymbol{x}\|_\infty = L/2\}$, a rectangle of length L and width W, which is rotated by an angle θ can be described with the transformation matrix $\boldsymbol{A} = \boldsymbol{R}(\theta)\boldsymbol{D}$, where $\boldsymbol{R}$ is the rotation matrix, and $\boldsymbol{D} = \text{diag}(1, L/W)$.

3.5 Experiments

In this section, we perform experiments solving inverse kinematics problems, motion planning and MPC for a task with hybrid dynamics, and motion planning for rectangular obstacle avoidance. The motivation behind these experiments is to show that: 1. the proposed way of solving these robotics problem can be unconventionally faster than the second-order methods such as iLQR and 2. exploiting projections whenever we can instead of leaving the constraints for the solver to treat them as generic constraints increases the performance significantly.

3.5.1 Constrained inverse kinematics

A constrained inverse kinematics problem can be described in many ways using projections. One typical way is to find a $\boldsymbol{q} \in \mathcal{C}_{\boldsymbol{q}}$ that minimizes a cost to be away from a given initial configuration $\boldsymbol{q}_0$ while respecting general constraints $\boldsymbol{h}(\boldsymbol{q}) = \boldsymbol{0}$ and projection constraints $\boldsymbol{f}(\boldsymbol{q}) \in \mathcal{C}_{\boldsymbol{x}}$

$$\begin{aligned} \min_{\boldsymbol{q} \in \mathcal{C}_{\boldsymbol{q}}} \quad & \|\boldsymbol{q} - \boldsymbol{q}_0\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{q}) = \boldsymbol{0}, \\ & \boldsymbol{f}(\boldsymbol{q}) \in \mathcal{C}_{\boldsymbol{x}}, \end{aligned} \tag{3.14}$$

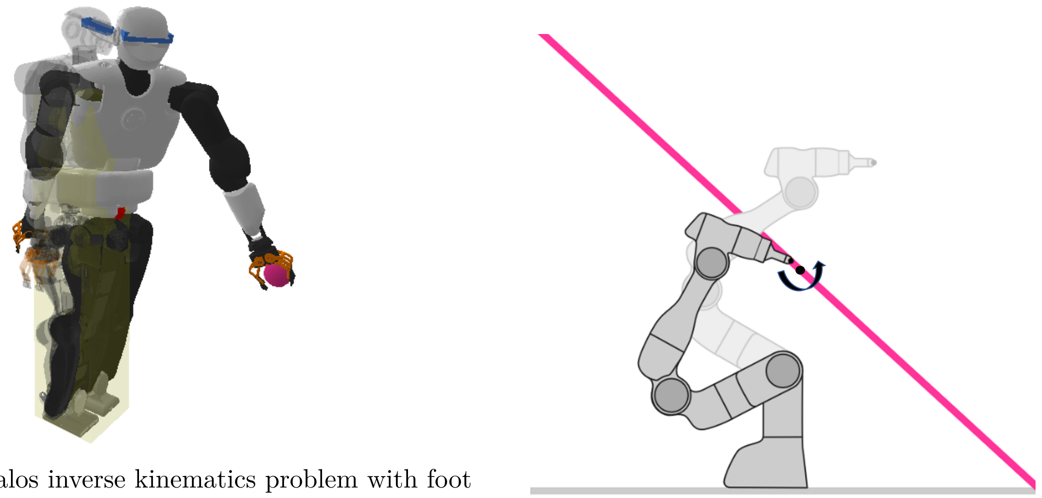
where $\boldsymbol{f}(\cdot)$ can represent entities such as the end-effector pose or the center of mass for which the constraints are easier to be expressed as projections onto $\mathcal{C}_{\boldsymbol{x}}$, and $\mathcal{C}_{\boldsymbol{q}}$ can represent the configuration space within the joint limits. Figure 3.1 shows 3DoF planar manipulator with $\boldsymbol{f}(\cdot)$ representing the end-effector position and $\mathcal{C}_{\boldsymbol{x}}$ denoting (a) $\mathcal{C}_{\boldsymbol{x}} = \{\boldsymbol{x} \mid \boldsymbol{x}{=}\boldsymbol{x}_d\}$, (b) $\mathcal{C}_{\boldsymbol{x}} = \{\boldsymbol{x} \mid \boldsymbol{a}^\top \boldsymbol{x} + b{=}0\}$, (c)$\mathcal{C}_{\boldsymbol{x}} = \{\boldsymbol{x} \mid \ \leq r_i^2 \leq \|\boldsymbol{x} - \boldsymbol{x}_d\|_2^2 \leq r_o^2\}$ and (d) $\mathcal{C}_{\boldsymbol{x}} = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}_d\|_{\infty, \boldsymbol{W}} \leq L\}$. We applied ALSPG algorithm iteratively to obtain a reactive control loop and we implemented it on an interactive webpage running python. The reader can refer to the provided website[3] for trying the code.

Talos IK: We tested our algorithm on a high dimensional (32 DoF) inverse kinematics problem of TALOS robot subject to constraints i) center of mass inside a box, ii) end-effector constrained to lie inside a sphere and iii) foot position and orientations are given. We compared two versions of ALSPG algorithm 1) by casting these constraints as projections onto $\mathcal{C}_{\boldsymbol{x}}$, 2) by keeping all the constraints inside the function $\boldsymbol{h}(\cdot)$ to see direct advantages of exploiting projections in ALSPG. We ran the algorithm from 1000 different random initial configurations for both cases and compared the number of function and Jacobian evaluations, n_f and n_j. For the case 1), we obtained $n_f{=}897.64 \pm 82.84$ and $n_j{=}883.44 \pm 81.7$, while for the case 2), we obtained $n_f{=}6459.4 \pm 3756.8$ and $n_j{=}3791.79 \pm 1061.05$.

Robust IK: In this experiment, we would like to achieve a task of reaching and staying in the half-space under a plane whose slope is stochastic because, for example, of the

---

[3] https://hgirgin.github.io/IK_SPG.html



(a) Talos inverse kinematics problem with foot pose, center of mass stability (red point inside yellow rectangular prism) and end-effector inside a (pink) sphere constraints.

(b) Robust inverse kinematics solution with $\mathcal{C}_{\boldsymbol{p}} = \{\boldsymbol{p} \mid \boldsymbol{\mu}^\top \boldsymbol{p} + \Psi^{-1}(\eta) \|\boldsymbol{\Sigma}^{\frac{1}{2}} \boldsymbol{p}\|_2 \leq 0\}$

Figure 3.3: Illustration of inverse kinematics problems solved with the proposed algorithm.

uncertainties in the measurements of the vision system. The constraint can be written as $\boldsymbol{a}^\top \boldsymbol{f}(\boldsymbol{q}) \leq 0$, where $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We can transform it into a chance-constraint to provide some safety guarantees in a probabilistic manner. The idea is to find a joint configuration $\boldsymbol{q}$ such that it will stay under a stochastic hyperplane with a probability of $\eta \geq 0.5$. Using Appendix A.2.5, this inequality can be written as a second-order cone constraint wrt $\boldsymbol{f}(\boldsymbol{q})$ as $\boldsymbol{\mu}^\top \boldsymbol{f}(\boldsymbol{q}) + \Psi^{-1}(\eta) \|\boldsymbol{\Sigma}^{\frac{1}{2}} \boldsymbol{f}(\boldsymbol{q})\|_2 \leq 0$, where $\Psi(\cdot)$ is the cumulative distribution function of zero mean unit variance Gaussian variable. Defining $\boldsymbol{g}(\boldsymbol{q}) = \begin{bmatrix} (\boldsymbol{\Sigma}^{\frac{1}{2}} f(\boldsymbol{q}))^\top & \boldsymbol{\mu}^\top \boldsymbol{f}(\boldsymbol{q}) \end{bmatrix}^\top$, the optimization problem can then be defined as

$$\begin{aligned} \min_{\boldsymbol{q} \in \mathcal{C}_{\boldsymbol{q}}} \quad & \|\boldsymbol{q} - \boldsymbol{q}_0\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{q}) \in \mathcal{C}_{\text{SOC}} \end{aligned} \tag{3.15}$$

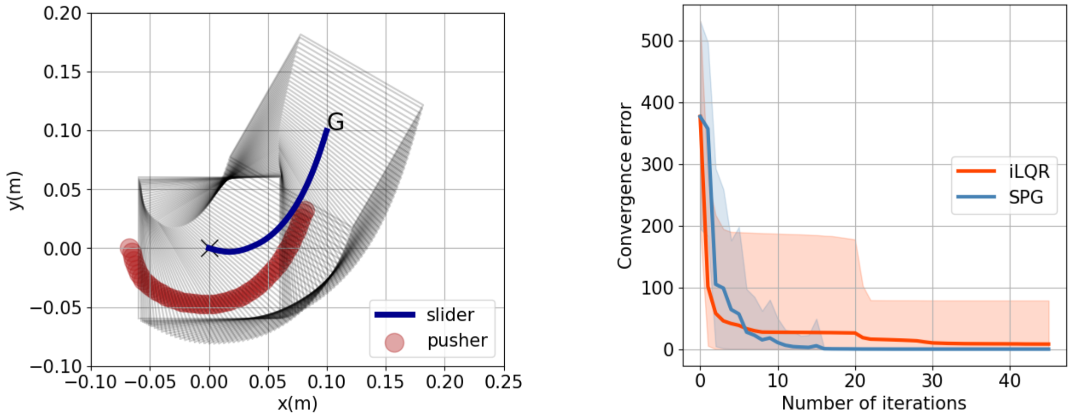
which can be solved efficiently without using second-order cone (SOC) gradients using the algorithm proposed in this chapter. We tested the algorithm on a 3DoF robot shown in Figure 3.3b by optimizing for a joint configuration for a probability of $\eta = 0.8$ and then computing continuously the constraint violation for the last 1000 times by sampling a line slope from the given distribution. We obtained a constraint violation percentage of around 80% as expected.

3.5.2 Motion planning and MPC on planar push

Non-prehensile manipulation has been widely studied as a challenging task for model-based planning and control, with the pusher-slider system as one of the most prominent

Table 3.2: Comparison of MPC with iLQR and ALSPG for planar push

	iLQR	ALSPG
Convergence time (s)	14.5 ± 1.25	2.93 ± 0.5
Number of function ev.	26689.5 ± 1830.5	6104.0 ± 1455.6
Number of jacobian ev.	225.9 ± 27.4	78.4 ± 8.5



(a) Pusher-slider system path optimized by the proposed algorithm to go from the state $(0, 0, 0)$ to $(0.1, 0.1, \pi/3)$. Optimal control solved with SPG results in a smooth path for the pusher-slider system.

(b) Convergence error mean and variance plot for iLQR and ALSPG motion planning algorithm for 10 different goal conditions starting from the same initial positions and control commands.

Figure 3.4: SPG algorithm applied to a pusher-slider system.

examples (see Figure 3.4). The reasons include hybrid dynamics with various interaction modes, underactuation and contact uncertainty. In this experiment, we study motion planning and MPC on this planar push system, without any constraints, to compare to a standard iLQR implementation. Motion planning convergence results for 10 different tasks are given for iLQR and ALSPG along with means and variances in Figure 3.4b. Although iLQR seems to converge to medium accuracy faster than ALSPG, because of the difficulties in the task dynamics, it seems to get stuck at local minima very easily. On the other hand, ALSPG seems to performing better in terms of variance and local minima. We applied MPC with iLQR and ALSPG with an horizon of 60 timesteps and stopped the MPC as soon as it reached the goal position with a desired precision. Table 3.2 shows this comparison in terms of convergence time (s), number of function evaluations and number of Jacobian evaluations. According to these findings, ALSPG seems to be perform better than a standard iLQR, even when there are no constraints in the problem.

Table 3.3: Comparison of motion planning for obstacle avoidance for three cases

	ALSPG-Proj.	SLSQP-Proj.	ALSPG-WoProj.
Convergence time (ms)	392.0 ± 66.2	679.0 ± 260.0	2780.0 ± 530.9
Number of function ev.	332.0 ± 60.6	438.4 ± 200.7	571.8 ± 109.9
Number of jacobian ev.	187.8 ± 34.6	389.2 ± 164.3	399.0 ± 93.9

3.5.3 Motion planning with obstacle avoidance

Obstacle avoidance problems are usually described using geometric constraints. In manipulators, capsules and spheres are used to represent the robot and the environment, such that the shortest distance computations and their gradients can be computed efficiently. In autonomous parking, obstacles and cars are usually described as 2D rectangular objects. In this experiment, we take a 2D double integrator point car reaching a target pose in the presence of rectangular obstacles (see Figure 3.5). We apply ALSPG algorithm with and without projections (ALSPG-Proj. and ALSPG-WoProj.) to illustrate the main advantages of having an explicit projection function over direct constraints. The main difference is without projections, the solvers need to compute the gradient of the constraints, whereas with projections, this is not necessary. In order to understand the differences between a first-order and a second-order methods, we also compared ALSPG-Proj to AL-SLSQP with projections (SLSQP-Proj.), which is the same algorithm except the subproblem is solved by a second-order solver SLSQP from Scipy. The objective function is $c(\boldsymbol{x}, \boldsymbol{u}) = 10^{-1} \|\boldsymbol{x}_T - \boldsymbol{x}_T^{\mathrm{G}}\|_2^2 + 10^{-4} \|\boldsymbol{u}_T\|_2^2$. We performed 5 experiments each with different settings of 4 rectangular obstacles and compared the convergence properties. The results are given in Table 3.3. The comparison between ALSPG-Proj. and ALSPG-WoProj. reports a clear advantage of using projections instead of plain constraints in the convergence properties. Although the convergence time comparison is not necessarily fair for SPG implementations as the SLSQP solver calls C++ functions, the comparison of ALSPG-Proj. and SLSQP-Proj. shows that ALSPG-Proj. still achieves lower convergence time.

3.5.4 MPC on Franka Emika

We tested ALSPG algorithm on an MPC problem of tracking an object with box constraints on the end-effector position of a Franka Emika robot (see Figure 3.6). The Aruco marker on the object is tracked by a camera held by another robot. In this experiment, the goal is to show the real-time applicability of the proposed algorithm for a constrained problem in the presence of disturbances. In Figure 3.7, the error of the constraints and the objective function is given for a 1 min. time period of MPC with a short-horizon of 50 timesteps. Between 20-30s and s, the robot is disturbed by the user thanks to the compliant torque controller run on the robot. We can see that the algorithm drives smoothly the error to zero. The accompanying video can be seen in the

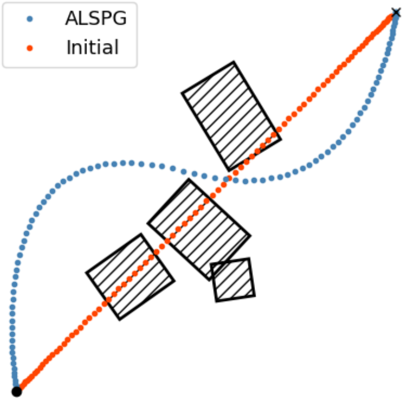


Figure 3.5: Motion planning problem in the presence of 4 scaled and rotated rectangular obstacles.

provided website.

3.6 Conclusion

In this chapter, we presented a fast first-order constrained optimization framework based on geometric projections widely used in robotics problems ranging from inverse kinematics to motion planning. We show that many of the geometric constraints can be rewritten as a logical combination of geometric primitives onto which the projections admit analytical expressions. We build an augmented Lagrangian method with spectral projected gradient descent as subproblem solver for constrained optimal control. We demonstrate : i. the advantages of using projections when compared to setting up the geometric constraints as plain constraints with gradient information to the solvers; and ii. the advantages of using spectral projected gradient descent based motion planning compared to a standard second-order iLQR algorithm through experiments on different robotic tasks.

Sample-based MPC have been increasingly popular in recent years thanks to its fast practical implementations despite their lack of theoretical guarantees. In contrast, second-order methods for MPC require a lot of computational power but with somewhat better convergence guarantees. We argue that ALSPG, being already in between these two methodologies in terms of these properties, promises great future work to combine it with sample-based MPC to further increase its advantages of both sides.

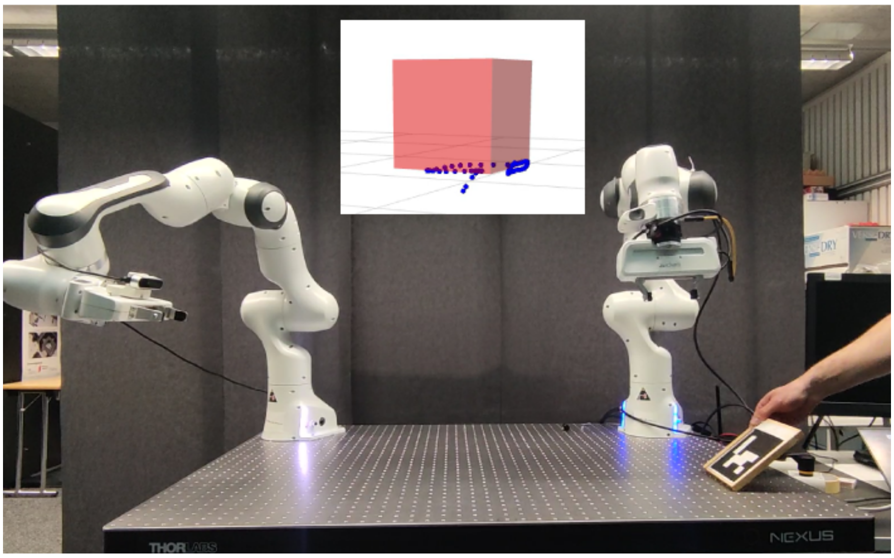


Figure 3.6: MPC setup for tracking an object subject to box constraints.

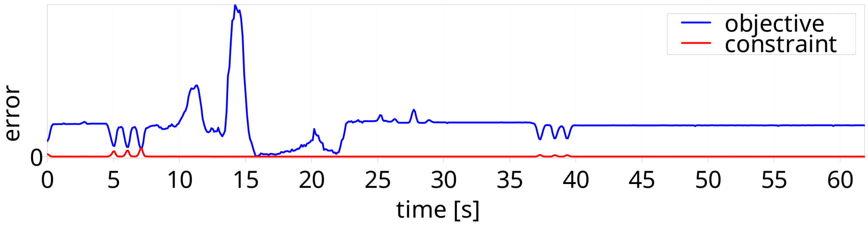


Figure 3.7: Error in the objective and the squared norm of the constraint value during 1 min execution of MPC on Franka Emika robot.

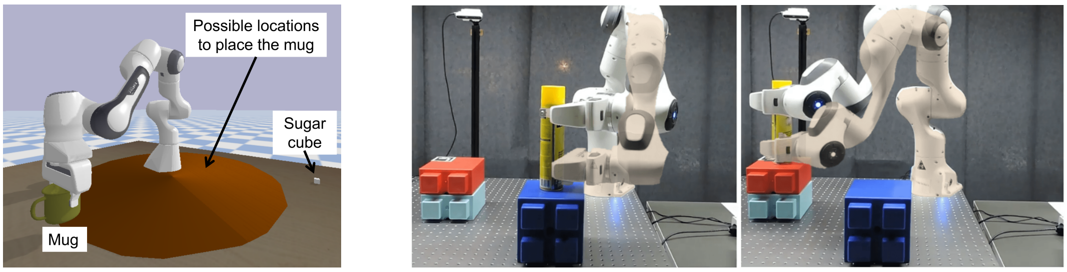
4 Robust Anticipatory Robot Skills with Memory

Optimal control can be found in a great variety of applications from economics [67] to engineering problems such as energy management [68] and robotics [69, 70]. It consists of determining optimal actions in problems following a known forward model that describe state changes in time when actions are applied.

In robotics, optimal control has been applied in many applications such as biped walking generation [71, 72], centroidal dynamics trajectory [73, 74] and whole-body motion planning [75]. These works often consider the solution of optimal trajectories of states and actions as a plan over a horizon, which is then tracked with lower level feedback control mechanisms. As the forward models that define these actions are not perfect representations of the real physical movements, feedback control is designed to achieve the planned motion even in the presence of noise, delays and unpredictable perturbations in the environment.

Among others, model predictive control (MPC) [76] is a powerful feedback mechanism in optimal control, that became a key methodology to control complex dynamical systems such as humanoids [77]. It allows to compute the optimal plan over a short receding horizon, apply the first few control commands until a new plan is recomputed using the current state of the robot. However, due to scaling issues, MPC approaches still have many challenges to be solved for high dimensional systems, for longer receding horizons, and for high frequency control.

Linear quadratic tracking (LQT) [78] and iterative linear quadratic regulator (iLQR) [2] and their variants are increasingly used in MPC frameworks for controlling high dimensional robotic systems such as humanoids and manipulators [4, 79, 77, 80]. In particular, dynamic programming solutions for these methods are often computationally less intensive than other trajectory optimization methods, with the additional benefit of outputting a full state feedback controller with gains over the horizon. While some works exploit these gains in real-time applications, other works use only the feedforward nominal solutions in MPC. The feedback controller structure in iLQR enables the feedback gains



(a) Mug-sugar cube scenario exploiting task precisions

(b) Pick-and-place task exploiting object affordances. The same controller results in two different grasping locations.

Figure 4.1: (a) The robot decides where to place the object according to the initial position and with the anticipation to pick up the sugar cube and put it inside the cup. (b) Pick-and-place task with the Franka Emika robot deciding autonomously to grasp the object from different parts, as an adaptation to different initial configurations. The memory feedback property allows the robot to remember where it placed the mug and grasped the object, so that it can place it accordingly.

to react to the current state of the system while anticipating the future states. While this is enough in many applications, they can not succeed in robotic applications where the robot needs to remember what it has done before to react and replan, since it does not have the notion of memory of the past states. The latter requires a controller which gets feedback from both the past states and the current state to find optimal control actions. For example, a robot grasping an object from its different parts needs to remember where it grasped the object to place it without collision as illustrated in Figure 4.1b.

Recently, System Level Synthesis (SLS) [21] was proposed as an optimal controller reacting not only to the current state but also to the past states of the system through closed-loop mappings optimization. This framework has been proven to be useful in distributed control systems with uncertain dynamical systems. Such a controller with memory could be exploited in tasks requiring correlations between states at different timesteps, as opposed to a controller without memory such as LQR controller. However, the SLS framework has a couple of limitations to be used in robotics applications requiring a memory of states with sparse cost functions and nonlinear dynamics.

In this chapter, we investigate how optimal control framework can produce anticipatory and reactive robot skills with a memory of the states to produce smart behaviors that can exploit task precisions and object affordances. We propose to achieve our goal with the following extensions to the SLS framework: 1) by adding a feedforward part to the controller enabling trajectory and viapoint tracking, 2) by extending the approach to nonquadratic costs and nonlinear systems using Newton method optimization; and 3) by providing fast adaptation and replanning strategies.

This chapter is organized as follows. First, Section 4.1 presents the related work in controllers with memory and SLS controllers, mostly within the domain of robotics. In

Section 4.2, we introduce the SLS framework and explain how LQR is a specific case of SLS. Then, in Section 4.3, we show our proposed extensions over SLS that enables its application in robotics and the proposed adaptation and replanning strategies when the robot encounters new situations. In Section 4.4, we showcase our approach with two scenarios exploiting task precisions and object affordances in pick-and-place tasks in a simulated and a real environment with a 7-axis Franka Emika robot. Finally, we conclude the work and discuss potential impacts of the work in Section 4.5.

Publication Note

The material presented in this chapter is adapted from the following publications:

- Girgin, H., Jankowski, J., and Calinon, S. (2023). Reactive anticipatory robot skills with memory. In Robotics Research (pp. 436-451). Cham: Springer Nature Switzerland.

4.1 Related Work

Feedback controllers that can act on the history of states are investigated in robotics mainly via reinforcement learning algorithms by changing the structure of the policy, especially with recurrent neural networks (RNN) [81]. In [82], the authors propose to learn stable bipedal locomotion with an RNN policy, where the memory serves as a model of the physical parameters of the task. In [83], a deep learning architecture for learning a policy that can remember some important information from the past observations is developed by augmenting the policy state with a continuous memory state. The authors showed that their guided policy search algorithm could solve a peg sorting task with a manipulator which needs to remember the target hole position given at the beginning of the training and a plate and bottle placing task where the robot needs to remember which object it is holding to determine the orientation.

System level synthesis (SLS) has been developed in a collection of works that are summarized in [21]. It provides a novel perspective on robust optimal control design by optimizing over the closed-loop mappings of the system, instead of directly optimizing the controller. The authors showed how SLS can be exploited in the domain of large-scale distributed optimal control and robust optimal control.

In [84], SLS has been exploited to learn unknown dynamics for safe control with LQR, including robust safety guarantees in the state and input constraints. In [85], perception based SLS controllers are designed for autonomous control of vehicles learning linear time invariant dynamical systems from image data. In [86], robust perception-based SLS controller is applied for the safe control of a quadrotor. The work in [87] gives necessary and sufficient conditions for the existence of SLS controller for nonlinear systems and [88] exploits these ideas by designing a nonlinear controller for a constrained LQR problem by blending several linear controllers.

4.2 Background

This section follows [21] to present the SLS framework for regulation problems. A linear-time-varying (LTV) system can be written as $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{w}_t, \forall t{=}\{0,\ldots,T\}$, where $\boldsymbol{x}_t \in \mathbb{R}^m$ is the state, $\boldsymbol{u}_t \in \mathbb{R}^n$ is the control input, and $\boldsymbol{w}_t \in \mathbb{R}^m$ is an exogenous disturbance term. By stacking these vectors for each time-step t, we define $\boldsymbol{x}{=}\begin{bmatrix}\boldsymbol{x}_0^\top & \boldsymbol{x}_1^\top & \ldots & \boldsymbol{x}_T^\top\end{bmatrix}$, $\boldsymbol{u}{=}\begin{bmatrix}\boldsymbol{u}_0^\top & \boldsymbol{u}_1^\top & \ldots & \boldsymbol{u}_T^\top\end{bmatrix}$ and $\boldsymbol{w}{=}\begin{bmatrix}\boldsymbol{x}_0^\top & \boldsymbol{w}_0^\top & \boldsymbol{w}_1^\top & \ldots & \boldsymbol{w}_{T-1}^\top\end{bmatrix}$, which allows us to express the dynamics as

$$\boldsymbol{x} = \boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{u} + \boldsymbol{w}, \tag{4.1}$$

where $\boldsymbol{Z}$ is a delaying operator with identity matrices along its first block sub-diagonal and zeros elsewhere, $\boldsymbol{A}_d{=}\,\mathrm{blkdiag}\,(\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_T)$ and $\boldsymbol{B}_d{=}\,\mathrm{blkdiag}(\boldsymbol{B}_0, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_T)$. In this work, we consider the stacked disturbance as $\boldsymbol{w}{\sim}\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)$, where $\boldsymbol{\Sigma}_w{=}\mathrm{blkdiag}(\boldsymbol{\Sigma}_{x_0}, \boldsymbol{\Sigma}_{\mathrm{noise}})$.

We assume a controller of the form $\boldsymbol{u}{=}\boldsymbol{K}\boldsymbol{x}$, where $\boldsymbol{K}$ is a lower block triangular matrix. Note that this controller is more advanced than the controller found by linear quadratic regulator (LQR). In fact, the former includes the latter at its block-diagonal elements while the off-block-diagonal elements act on the history of the states. Inserting the controller definition into (4.1), we get $\boldsymbol{x}{=}\boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{K}\boldsymbol{x} + \boldsymbol{w}$, which results in the closed-loop responses

$$\boldsymbol{x} = \boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{w}, \quad \boldsymbol{\Phi}_{\boldsymbol{x}} = \Big(\boldsymbol{I} - \boldsymbol{Z}(\boldsymbol{A}_d + \boldsymbol{B}_d\boldsymbol{K})\Big)^{-1}, \tag{4.2}$$

$$\boldsymbol{u} = \boldsymbol{\Phi}_{\boldsymbol{u}}\boldsymbol{w}, \quad \boldsymbol{\Phi}_{\boldsymbol{u}} = \boldsymbol{K}\Big(\boldsymbol{I} - \boldsymbol{Z}(\boldsymbol{A}_d + \boldsymbol{B}_d\boldsymbol{K})\Big)^{-1}, \tag{4.3}$$

where $\{\boldsymbol{\Phi}_{\boldsymbol{x}}, \boldsymbol{\Phi}_{\boldsymbol{u}}\}$ describe the closed loop system responses from the exogenous disturbance $\boldsymbol{w}$ to the state $\boldsymbol{x}$ and control input $\boldsymbol{u}$, respectively. SLS optimizes directly over these system responses, instead of the controller map $\boldsymbol{K}$ as was done by the dynamic programming solutions of LQR. As the controller $\boldsymbol{K}$ is a block lower triangular matrix, these maps are also lower block triangular matrices as follows:

$$\boldsymbol{\Phi_x} = \begin{bmatrix} \boldsymbol{\Phi}_x^{0,0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}_x^{1,0} & \boldsymbol{\Phi}_x^{1,1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{\Phi}_x^{T,0} & \boldsymbol{\Phi}_x^{T,1} & \boldsymbol{\Phi}_x^{T,2} & \cdots & \boldsymbol{\Phi}_x^{T,T-1} & \boldsymbol{\Phi}_x^{T,T} \end{bmatrix},$$

$$\boldsymbol{\Phi_u} = \begin{bmatrix} \boldsymbol{\Phi}_u^{0,0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}_u^{1,0} & \boldsymbol{\Phi}_u^{1,1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{\Phi}_u^{T,0} & \boldsymbol{\Phi}_u^{T,1} & \boldsymbol{\Phi}_u^{T,2} & \cdots & \boldsymbol{\Phi}_u^{T,T-1} & \boldsymbol{\Phi}_u^{T,T} \end{bmatrix}.$$

By inserting (4.2) and (4.3) into the dynamics equation (4.1), we obtain

$$\begin{aligned} & \boldsymbol{\Phi_x w} = \boldsymbol{Z A}_d \boldsymbol{\Phi_x w} + \boldsymbol{Z B}_d \boldsymbol{\Phi_u w} + \boldsymbol{w}, \\ \implies & \boldsymbol{\Phi_x} = \boldsymbol{Z A}_d \boldsymbol{\Phi_x} + \boldsymbol{Z B}_d \boldsymbol{\Phi_u} + \boldsymbol{I} = \boldsymbol{S_x} + \boldsymbol{S_u \Phi_u}, \end{aligned} \tag{4.4}$$

where the implication is because we want this pair of system responses to remain independent of the noise in the system for every initial state, and $\boldsymbol{S_x}{=}(\boldsymbol{I} - \boldsymbol{Z A}_d)^{-1}$, $\boldsymbol{S_u}{=}\boldsymbol{S_x Z B}_d$. Note that these matrices can also be constructed recursively as explained in Section 2.2.1. The objective of SLS is to optimize directly over these lower block triangular system responses $\{\boldsymbol{\Phi_x}, \boldsymbol{\Phi_u}\}$ so that there exists a linear controller $\boldsymbol{K}$ such that $\boldsymbol{K}{=}\boldsymbol{\Phi_u \Phi_x}^{-1}$ and (4.4) holds as outlined by the Theorem 2.1 of [21].

Linear quadratic regulator: We consider now a linear quadratic regulator problem with random noise in the process and with a random initial state as

$$\begin{aligned} \min_{\boldsymbol{x}_t, \boldsymbol{u}_t} \quad & \textstyle\sum_{t=0}^{T} \mathbb{E}[\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t] \\ \text{s.t.} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{A x}_t + \boldsymbol{B u}_t + \boldsymbol{w}_t. \end{aligned} \tag{4.5}$$

Using the stacked vectors $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{w}$ and the block diagonal matrices $\boldsymbol{Q}{=}\text{blkdiag}(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R}{=}\text{blkdiag}(\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_T)$, this problem can be recast as optimization over system responses as

$$\begin{aligned} \min_{\boldsymbol{\Phi_x}, \boldsymbol{\Phi_u}} \quad & \mathbb{E}[\|\boldsymbol{\Phi_x w}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{\Phi_u w}\|_{\boldsymbol{R}}^2] \\ \text{s.t.} \quad & \boldsymbol{\Phi_x} = \boldsymbol{S}_x + \boldsymbol{S}_u \boldsymbol{\Phi_u}, \\ & \boldsymbol{\Phi_x}, \boldsymbol{\Phi_u} \in \mathcal{L} \end{aligned} \tag{4.6}$$

where $\mathcal{L}$ represents the space of lower block triangular matrices. Solving (4.6), we obtain a feedback controller achieving the desired system responses. It is possible to show that we can remove the expectation and treat the cost function as a random valued cost function. This allows us to solve the problem independently of the value of $\boldsymbol{w}$. Therefore, in the remainder of this chapter, we take the cost functions as random valued cost functions instead of their expectations.

Note that because of the constraint to lie on the space $\mathcal{L}$, this problem cannot be solved directly as we would solve a simple LQR problem. Instead, we make use of the column separability of the objective function and the equality constraint representing the dynamics model to separate the problem into T independent subproblems containing only the nonzero part of each block column of the system responses as in [21].

The controller found by solving (4.6) can be applied directly to problems with linear forward models and quadratic costs, if the task is to regulate the state to a set-point or to a reference trajectory, which is already dynamically feasible by the plant. In fact, in these cases, one can transform the forward state dynamics to forward error dynamics and still exploit SLS to solve such tasks. However, in most of the robotics applications, the reference trajectory is composed of sparse viapoints and/or the system is nonlinear and/or the cost is nonquadratic. In the next section, we show how to alleviate these problems as part of our contributions.

4.3 Methods

In this section, we show that we can solve SLS problems for tracking analytically and how to solve them for nonlinear systems and nonquadratic costs. In Section 4.3.1, we extend the SLS method to *extended system level synthesis* (eSLS) by adding a feedforward part that describes the desired states and the desired control commands. Then, in Section 4.3.2, we exploit eSLS to solve the problems with nonquadratic cost and nonlinear dynamics, resulting in an *iterative system level synthesis* (iSLS) approach, which greatly extends the domain of applications of the standard SLS.

4.3.1 Extended system level synthesis (eSLS)

The closed loop responses in Equation (4.2) and Equation (4.3) cannot represent the closed-loop dynamics well when the problem is to track a desired state $\boldsymbol{x}_{(d,t)}$ and a desired control command $\boldsymbol{u}_{(d,t)}$ by minimizing $J{=}\mathbb{E}[\|\boldsymbol{x}-\boldsymbol{x}_d\|^2_{\boldsymbol{Q}}+\|\boldsymbol{u}-\boldsymbol{u}_d\|^2_{\boldsymbol{R}}]$ In this section, we propose a new closed-loop response model that can explicitly encode the desired states and the control commands by extending the linear feedback controller in SLS to include also a feedforward term as $\boldsymbol{u}=\boldsymbol{K}\boldsymbol{x}+\boldsymbol{k}$. Inserting this into the dynamics equation Equation (4.1), we obtain $\boldsymbol{x}=\boldsymbol{Z}\boldsymbol{A}_d\boldsymbol{x}+\boldsymbol{Z}\boldsymbol{B}_d(\boldsymbol{K}\boldsymbol{x}+\boldsymbol{k})+\boldsymbol{w}$, which results in the closed-loop responses

$$\boldsymbol{x}=\boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{x}},\quad \boldsymbol{d}_{\boldsymbol{x}}=\boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{Z}\boldsymbol{B}_d\boldsymbol{k}, \tag{4.7}$$

$$\boldsymbol{u}=\boldsymbol{\Phi}_{\boldsymbol{u}}\boldsymbol{w}+\boldsymbol{d}_{\boldsymbol{u}},\quad \boldsymbol{d}_{\boldsymbol{u}}{=}\boldsymbol{K}\boldsymbol{d}_{\boldsymbol{x}}+\boldsymbol{k}, \tag{4.8}$$

where $\boldsymbol{\Phi}_{\boldsymbol{x}}$ and $\boldsymbol{\Phi}_{\boldsymbol{u}}$ are defined as in Equation (4.2) and Equation (4.3). When we insert $\boldsymbol{x}$ and $\boldsymbol{u}$ in Equation (4.7) and Equation (4.8) into the dynamics equation Equation (4.1), we get the dynamics constraints on our optimization variables from $\boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{w} + \boldsymbol{S}_{\boldsymbol{u}}(\boldsymbol{\Phi}_{\boldsymbol{u}}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{u}})$, which is satisfied for any noise in the system by Equation (4.4) and $\boldsymbol{d}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{d}_{\boldsymbol{u}}$. This can be used to show that $\boldsymbol{k} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})\boldsymbol{d}_{\boldsymbol{u}}$. By a few manipulations of equations, one can rewrite the controller into the more interpretable form of $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{x} + \boldsymbol{k} = \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{d}_{\boldsymbol{x}}) + \boldsymbol{d}_{\boldsymbol{u}}$, where $\boldsymbol{d}_{\boldsymbol{x}}$ and $\boldsymbol{d}_{\boldsymbol{u}}$ can be interpreted as the planned trajectory of states and control commands, respectively, assuming zero noise and zero initial state. The feedback part drives the system to a new plan when there is noise, perturbations or nonzero initial state.

The final convex optimization problem becomes

$$\begin{aligned} \min_{\substack{\boldsymbol{\Phi}_{\boldsymbol{x}}, \boldsymbol{\Phi}_{\boldsymbol{u}}, \\ \boldsymbol{d}_x, \boldsymbol{d}_u}} \quad & \|\boldsymbol{\Phi}_{\boldsymbol{x}}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{x}} - \boldsymbol{x}_d\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{\Phi}_{\boldsymbol{u}}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{u}_d\|_{\boldsymbol{R}}^2 \\ \text{s.t.} \quad & \boldsymbol{\Phi}_{\boldsymbol{x}} = \boldsymbol{S}_x + \boldsymbol{S}_u\boldsymbol{\Phi}_{\boldsymbol{u}}, \\ & \boldsymbol{d}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{d}_{\boldsymbol{u}}, \\ & \boldsymbol{\Phi}_{\boldsymbol{x}}, \boldsymbol{\Phi}_{\boldsymbol{u}} \in \mathcal{L} \end{aligned} \tag{4.9}$$

which can be solved analytically for the optimization variables. For the sake of readability, we omit the details on the derivation and instead give the final algorithm in Algorithm 4.

Algorithm 4: Extended System Level Synthesis

```
Solve for feedforward terms
    $\boldsymbol{d}_{\boldsymbol{u}} = (\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1}(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{x}_d + \boldsymbol{R}\boldsymbol{u}_d),$
while $i < T$ do  Solve for feedback terms
    $\hat{\boldsymbol{\Phi}}_u^i = -(\boldsymbol{S}_{\boldsymbol{u}}^{i\top} \boldsymbol{Q}^i \boldsymbol{S}_{\boldsymbol{u}}^i + \boldsymbol{R}^i)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^{i\top} \boldsymbol{Q}^i \boldsymbol{S}_{\boldsymbol{x}}^i$
    $\hat{\boldsymbol{\Phi}}_x^i = \boldsymbol{S}_{\boldsymbol{x}}^i + \boldsymbol{S}_{\boldsymbol{u}}^i \hat{\boldsymbol{\Phi}}_u^i$
end
Compute the feedback and feedforward parts of the controller with $\boldsymbol{K} = \boldsymbol{\Phi}_{\boldsymbol{u}}\boldsymbol{\Phi}_x^{-1}$ ,
  $\boldsymbol{k} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S}_{\boldsymbol{u}})\boldsymbol{d}_{\boldsymbol{u}}$
```

4.3.2 Iterative system level synthesis (iSLS)

In this subsection, we consider the problem of system level synthesis for non-linear dynamical systems and non-quadratic cost functions. We perform a first order Taylor expansion of the dynamical system $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t$ around some nominal realization of the plant denoted as $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t, \boldsymbol{\mu}_{\boldsymbol{w}_t})$, namely, $\boldsymbol{x}_{t+1} \approx \boldsymbol{f}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \boldsymbol{\mu}_{\boldsymbol{w}_t} + \boldsymbol{A}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \boldsymbol{B}_t(\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t) + (\boldsymbol{w}_t - \boldsymbol{\mu}_{\boldsymbol{w}_t})$ with Jacobian matrices $\{\boldsymbol{A}_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \boldsymbol{B}_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t}|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}\}$. Using the notations $\hat{\boldsymbol{x}}_{t+1}{=}\boldsymbol{f}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \boldsymbol{\mu}_{w_t}$, $\Delta\boldsymbol{x}_t{=}\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$, $\Delta\boldsymbol{u}_t{=}\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t$, the linearized dynamics model can be rewritten as

$$\begin{aligned} \Delta\boldsymbol{x}_{t+1} &= \boldsymbol{A}_t\Delta\boldsymbol{x}_t + \boldsymbol{B}_t\Delta\boldsymbol{u}_t + \Delta\boldsymbol{w}_t, \forall t, \\ \iff \Delta\boldsymbol{x} &= \boldsymbol{Z}\boldsymbol{A}_d\Delta\boldsymbol{x} + \boldsymbol{Z}\boldsymbol{B}_d\Delta\boldsymbol{u} + \Delta\boldsymbol{w}, \end{aligned}$$

where the second line is the stacked form of the linearized dynamics model. We assume a controller of the form $\Delta\boldsymbol{u} = \boldsymbol{K}\Delta\boldsymbol{x} + \boldsymbol{k}$ and follow the same procedure described in the previous section to write the closed-loop responses as $\Delta\boldsymbol{x} = \boldsymbol{\Phi}_{\boldsymbol{x}}\Delta\boldsymbol{w} + \boldsymbol{d}_x$ and $\Delta\boldsymbol{u} = \boldsymbol{\Phi}_{\boldsymbol{u}}\Delta\boldsymbol{w} + \boldsymbol{d}_u$, where the variables $\boldsymbol{\Phi}_{\boldsymbol{x}}, \boldsymbol{\Phi}_{\boldsymbol{u}}, \boldsymbol{d}_x, \boldsymbol{d}_u$ satisfy the same constraints in (4.9) and (4.4) with respect to the linearized dynamics model.

The cost function $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ can be approximated by a second order Taylor expansion around $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, namely, $c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \boldsymbol{c}_{\boldsymbol{x}_t}^\top\Delta\boldsymbol{x}_t + \boldsymbol{c}_{\boldsymbol{u}_t}^\top\Delta\boldsymbol{u}_t + \frac{1}{2}\Delta\boldsymbol{x}_t^\top\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}\Delta\boldsymbol{x}_t + \frac{1}{2}\Delta\boldsymbol{u}_t^\top\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{u}_t}\Delta\boldsymbol{u}_t + \Delta\boldsymbol{u}_t^\top\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{x}_t}\Delta\boldsymbol{x}_t$ with $\boldsymbol{c}_{\boldsymbol{x}_t}{=}\frac{\partial c}{\partial \boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}$, $\boldsymbol{c}_{\boldsymbol{u}_t}{=}\frac{\partial c}{\partial \boldsymbol{u}_t}|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}$, $\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}{=}\frac{\partial^2 c}{\partial {\boldsymbol{x}_t}^2}|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}$, $\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{u}_t}{=}\frac{\partial^2 c}{\partial {\boldsymbol{u}_t}^2}|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}$, $\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{x}_t}{=}\frac{\partial^2 c}{\partial \boldsymbol{u}_t\boldsymbol{x}_t}|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}$. Using the notations $\boldsymbol{x}_{(d,t)}{=} - \boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}^{-1}\boldsymbol{c}_{\boldsymbol{x}_t}$, $\boldsymbol{u}_{(d,t)}{=} - \boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{u}_t}^{-1}\boldsymbol{c}_{\boldsymbol{u}_t}$ and assuming that $\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{x}_t}{=}\boldsymbol{0}$ and neglecting the constant terms, the quadratized cost function becomes

$$\begin{aligned} c(\boldsymbol{x}_t, \boldsymbol{u}_t) &= (\Delta\boldsymbol{x}_t - \boldsymbol{x}_{(d,t)})^\top\boldsymbol{C}_{\boldsymbol{x}_t\boldsymbol{x}_t}(\Delta\boldsymbol{x}_t - \boldsymbol{x}_{(d,t)}) + \\ &\quad (\Delta\boldsymbol{u}_t - \boldsymbol{u}_{(d,t)})^\top\boldsymbol{C}_{\boldsymbol{u}_t\boldsymbol{u}_t}(\Delta\boldsymbol{u}_t - \boldsymbol{u}_{(d,t)}), \qquad (4.10) \\ c(\boldsymbol{x}, \boldsymbol{u}) &= \|\Delta\boldsymbol{x} - \boldsymbol{x}_d\|^2_{\boldsymbol{C}_{\boldsymbol{x}\boldsymbol{x}}} + \|\Delta\boldsymbol{u} - \boldsymbol{u}_d\|^2_{\boldsymbol{C}_{\boldsymbol{u}\boldsymbol{u}}} \qquad (4.11) \end{aligned}$$

where the second equality is the stacked form of the quadratized cost function.

We propose to perform a Newton's step at each iteration by solving a tracking problem of the same form as (4.9) by changing the state and the control to $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{u}$. Thus, this results in the following convex optimization problem

$$\begin{aligned} \min_{\substack{\boldsymbol{\Phi}_{\boldsymbol{x}},\boldsymbol{\Phi}_{\boldsymbol{u}},\\ \boldsymbol{d}_x,\boldsymbol{d}_u}} \quad & \|\boldsymbol{\Phi}_{\boldsymbol{x}}\Delta\boldsymbol{w}{+}\boldsymbol{d}_x{-}\boldsymbol{x}_d\|^2_{\boldsymbol{C}_{\boldsymbol{x}\boldsymbol{x}}} + \|\boldsymbol{\Phi}_{\boldsymbol{u}}\Delta\boldsymbol{w}{+}\boldsymbol{d}_u{-}\boldsymbol{u}_d\|^2_{\boldsymbol{C}_{\boldsymbol{u}\boldsymbol{u}}} \\ \text{s.t.} \quad & \boldsymbol{\Phi}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{x}} + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{\Phi}_{\boldsymbol{u}}, \qquad (4.12) \\ & \boldsymbol{d}_x = \boldsymbol{S}_u\boldsymbol{d}_u, \\ & \boldsymbol{\Phi}_{\boldsymbol{x}}, \boldsymbol{\Phi}_{\boldsymbol{u}} \in \mathcal{L} \end{aligned}$$

which can be solved analytically the same way as eSLS. Newton optimization methods are known to be prone to overshoots, which can be handled via line search. We propose a line search algorithm based on the feedforward control term of our controller as was done by previously proposed iLQR algorithms [2]. Specifically, we define a variable α with a line search strategy with $\boldsymbol{d}_u^{k+1} = \alpha\boldsymbol{d}_u^k$. We accept this update if the trajectories found by these closed loop dynamics models decrease our actual cost function, otherwise we decrease α and re-assess. This corresponds to updating $\boldsymbol{k}$ in the same manner.

Algorithm 5: Iterative System Level Synthesis (iSLS)

```
Initialize the nominal state $\hat{\boldsymbol{x}}_t$ and control $\hat{\boldsymbol{u}}_t$ ;
Initialize the change in the cost $\Delta c$ ;
Set a threshold $\tau$ ;
while $|\Delta c| > \tau$ do  Solve iSLS
    Linearize the dynamics and quadratize the cost function around $\{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t\}_{t=0}^T$
      to find $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}_{\boldsymbol{xx}}$, $\boldsymbol{x}_d$ and $\boldsymbol{u}_d$ ;
    Solve (4.12) to find $\boldsymbol{K}$ and $\boldsymbol{k}$ ;
    Do line search to update $\boldsymbol{k}$ using the controller $\Delta\boldsymbol{u} = \boldsymbol{K}\Delta\boldsymbol{x} + \boldsymbol{k}$ and the
      dynamics model ;
    Update $\Delta c$.
end
```

Time correlations between states

The controller defined by SLS reacts not only to the current state error of the robot, but also to the history of previous states. Such a controller with memory can be exploited in tasks requiring correlations between states at different timesteps, as opposed to a controller without memory such as LQR controllers. An SLS controller can remember what it did in the previous part of the trajectory to use this information to better anticipate the future states and to successfully complete tasks where the future trajectory depends on past states. An example of such task is illustrated in Figure 4.1.

To achieve correlations between different timesteps, we make use of the off-block-diagonal elements of the precision matrix. Let us denote $\boldsymbol{x}_{t_1}$ and $\boldsymbol{x}_{t_2}$ the states at t_1 and t_2 that we want to correlate. The correlations that we consider in this work are in the form $\boldsymbol{C}\boldsymbol{x}_{t_1}+\boldsymbol{c}\sim\boldsymbol{x}_{t_2}$, where $\boldsymbol{C}$ and $\boldsymbol{c}$ are the coefficient matrix and the vector, respectively. We define the correlation cost $c(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2})$ as

$$\begin{aligned} c(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) &= (\boldsymbol{C}\boldsymbol{x}_{t_1} + \boldsymbol{c} - \boldsymbol{x}_{t_2})^\top \boldsymbol{Q}_\mathrm{c} (\boldsymbol{C}\boldsymbol{x}_{t_1} + \boldsymbol{c} - \boldsymbol{x}_{t_2}), \\ &= \boldsymbol{x}_{t_1}^\top \underbrace{\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c} \boldsymbol{C}}_{\boldsymbol{Q}_{t_1}} \boldsymbol{x}_{t_1} + (\boldsymbol{x}_{t_2} - \boldsymbol{c})^\top \underbrace{\boldsymbol{Q}_\mathrm{c}}_{\boldsymbol{Q}_{t_2}} (\boldsymbol{x}_{t_2} - \boldsymbol{c}) \\ + \boldsymbol{x}_{t_1}^\top & \underbrace{\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c}}_{-\boldsymbol{Q}_{t_1 t_2}} (\boldsymbol{x}_{t_2} - \boldsymbol{c}) - 2(\boldsymbol{x}_{t_2} - \boldsymbol{c})^\top \underbrace{\boldsymbol{Q}_\mathrm{c} \boldsymbol{C}}_{-\boldsymbol{Q}_{t_2 t_1}} \boldsymbol{x}_{t_1}, \\ &= \begin{bmatrix} \boldsymbol{x}_{t_1} \\ \boldsymbol{x}_{t_2} - \boldsymbol{c} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_{t_1} & \boldsymbol{Q}_{t_1 t_2} \\ \boldsymbol{Q}_{t_2 t_1} & \boldsymbol{Q}_{t_2} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{t_1} \\ \boldsymbol{x}_{t_2} - \boldsymbol{c} \end{bmatrix}. \end{aligned} \tag{4.13}$$

This means adding off-block-diagonal elements to a typical block-diagonal precision matrix $\boldsymbol{Q}$:=blkdiag$(\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_T)$, by setting $\boldsymbol{Q}(t_1, t_1) = \boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c} \boldsymbol{C}$, $\boldsymbol{Q}(t_2, t_2) = \boldsymbol{Q}_\mathrm{c}$, $\boldsymbol{Q}(t_1, t_2) = -\boldsymbol{C}^\top \boldsymbol{Q}_\mathrm{c}$ and $\boldsymbol{Q}(t_2, t_1) = -\boldsymbol{Q}_\mathrm{c} \boldsymbol{C}$, where $\boldsymbol{Q}(t_i, t_j)$ represents the precision matrix block corresponding to the timesteps i and j.

Adaptation to new reference states

We remark that while the system responses encode the information about the precision of the task only, the feedforward part of the responses, namely, $\boldsymbol{d_x}$ and $\boldsymbol{d_u}$, encode the desired states $\boldsymbol{x}_d$ and the desired control commands $\boldsymbol{u}_d$. Moreover, $\boldsymbol{d_x}$ and $\boldsymbol{d_u}$ are a linear function of $\boldsymbol{x}_d$ and $\boldsymbol{u}_d$ (see Algorithm 4), which makes them easily recomputable when one changes the desired states and control commands while keeping the precision matrices constant. Examples include trajectory tracking with the same precision throughout the horizon, a task where the robot needs to reach different viapoints and final goal tasks. At the controller level, this only corresponds to a change in the feedforward term $\boldsymbol{k}$, while the feedback part $\boldsymbol{K}$ stays the same. We have $\boldsymbol{k} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S_u})\boldsymbol{d_u}, = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S_u})(\boldsymbol{S_u}^\top\boldsymbol{Q}\boldsymbol{S_u} + \boldsymbol{R})^{-1}(\boldsymbol{S_u}^\top\boldsymbol{Q}\boldsymbol{x}_d + \boldsymbol{R}\boldsymbol{u}_d), = \boldsymbol{F_x}\boldsymbol{x}_d + \boldsymbol{F_u}\boldsymbol{u}_d$, where the terms $\boldsymbol{F_x}$=$(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S_u})(\boldsymbol{S_u}^\top\boldsymbol{Q}\boldsymbol{S_u} + \boldsymbol{R})^{-1}\boldsymbol{S_u}^\top\boldsymbol{Q}$ and $\boldsymbol{F_u}$=$(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{S_u})(\boldsymbol{S_u}^\top\boldsymbol{Q}\boldsymbol{S_u} + \boldsymbol{R})^{-1}\boldsymbol{R}$ can be computed offline and can be used when the desired state and control commands are changed to determine the new feedforward control term $\boldsymbol{k}$ only by matrix-vector multiplication operation along the changed time and dimensions.

In the eSLS formulation with linear dynamics and quadratic cost, the replanned motion and the controller are optimal, i.e., if one wants to optimize from scratch when the desired state is changed, then one ends up with the same solution. However, in iSLS, the replanned motion produces valid trajectories only in the vicinity of the optimal solution. We argue and show that this can still be exploited to produce trajectories which achieve the task successfully without any computation overhead of planning from scratch.

4.4 Experiments and Results

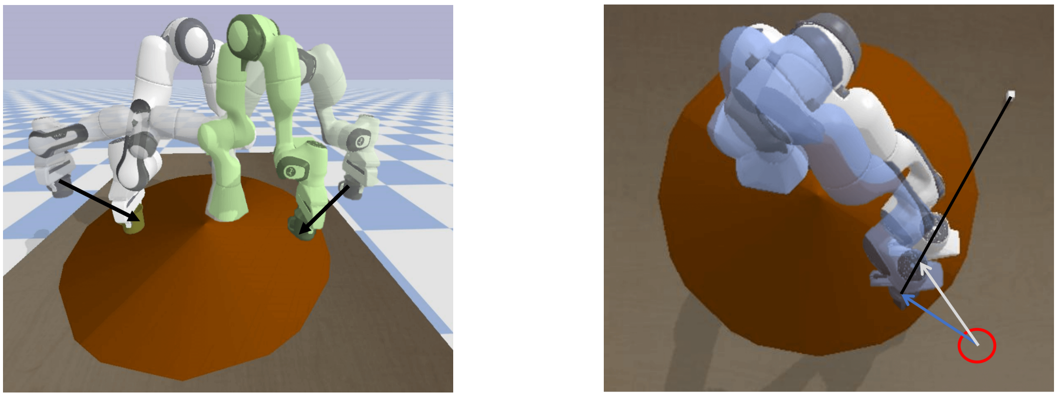
This section showcases the proposed approach with robotic tasks on a simulated and real environment with Franka Emika robot. The experiment videos for the real task can be found in the video[1].

4.4.1 Simulated task

We implemented a simulated robotic task exploiting task precisions with the help of the memory feedback. The robot, initially holding a coffee mug, needs to place it on the table within a certain range depicted as brown disk shape on the table (first viapoint, t=20), pick up a sugar cube depicted as a white cube (second viapoint, t=70) and drop it onto the mug (goal point, t=100) as illustrated in Figure 4.1a. Depending on the initial position of the robot or the perturbations in the environment, the robot can decide on

---

[1] The video can be accessed via https://hgirgin.github.io



(a) Adaptation to two different initial positions (transparent robots) by following the arrows.

(b) Comparison of the execution of MPC-LQT (blue robot) and eSLS (white robot) controllers

Figure 4.2: Simulated task where the robot, initially holding a coffee mug, needs to place it on the table within a certain range (brown disk), pick up a sugar cube (white cube) and drop it onto the mug.

different locations to place the mug. This location needs then to be remembered to drop the sugar cube from the correct location. We implemented eSLS with a double integrator dynamics with 100 timesteps to design a feedback controller on the task space and an inverse kinematics controller to track the reference trajectory generated online by the feedback controller. The cost function have three state cost components and a control cost as $c{=}\|\boldsymbol{x}_{20} - \boldsymbol{c}_d\|^2_{\boldsymbol{Q}_{20}} + \|\boldsymbol{x}_{70} - \boldsymbol{c}_s\|^2_{\boldsymbol{Q}_{70}} + \|\boldsymbol{x}_{20} - \boldsymbol{x}_{100}\|^2_{\boldsymbol{Q}_{20,100}} + 0.01\|\boldsymbol{u}\|$, where $\boldsymbol{c}_d$ and $\boldsymbol{c}_s$ are the center of the brown disk and the location of the sugar cube, respectively, and $\boldsymbol{Q}_{20}{=}\,\mathrm{diag}(10^3, 10^3, 10^5, 10^5, 10^5, 10^5)$, $\boldsymbol{Q}_{70}{=}10^5{\times}\boldsymbol{I}$ $\boldsymbol{Q}_{20,100}{=}\,\mathrm{diag}(10^5{\times}\boldsymbol{I}_3, \boldsymbol{0}_3)$. Figure 4.2a shows two examples of eSLS controller starting from two different positions and hence choosing two different locations to place the mug by anticipating that it will need to put the sugar cube inside the mug. These locations differ as there is a trade-off between the state and control costs. As the eSLS controller has memory, the robot can remember this location to accomplish the task.

As a baseline to compare, we chose to design an LQT controller solved by dynamic programming. Obviously, this controller have no information about the previous states and neither about the off-diagonal elements in the precision matrices, hence it can not achieve the task alone. A quick but not generalizable solution to this problem is to recompute LQT controller after the mug is placed on the table, almost as in MPC, but recomputing the solution only once. We call this controller MPC-LQT. We argue and show that this strategy is far from the optimal behavior because it eliminates the anticipatory and memory aspects of the controller. We tested MPC-LQT against our proposed framework with 10 different initial positions of the robot end-effector, each deciding on different locations of the mug. One such execution is shown in Figure 4.2b, with blue and white robots illustrating MPC-LQT and eSLS strategies, respectively.

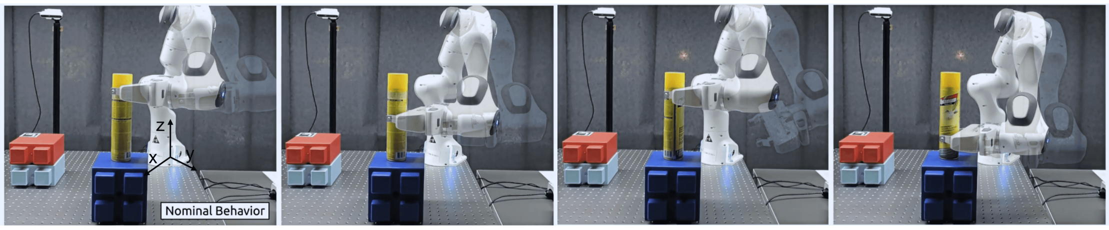


Figure 4.3: Adaptation of the robot to different initial configurations shown in transparent and the grasping locations shown in solid colors.

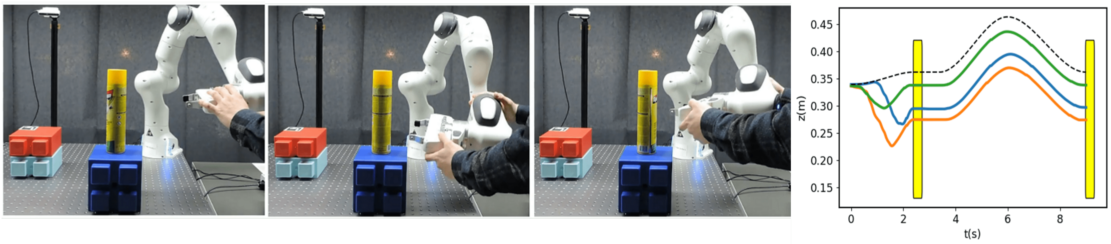


Figure 4.4: Testing of the reactivity of the controller to different physical perturbations, each resulting in successful completion of the task by adaption. The plot of z-axis position (m) in time (s) gathered from the robot perturbed before grasping the object. The curves of color green, orange and blue correspond to the first, second and third screenshots respectively, while the dashed black line corresponds to the nominal solution. The robot decides on-the-fly to grasp from different locations and remembers these locations to place the yellow object without colliding with the environment.

Each robot, starting from the same initial position depicted by a red circle, places the mug in different locations following the blue and white arrows, and picks up the sugar cube and places it inside the mug correctly. Even though successful, one can see from the geometry in the figure that the final path taken by the MPC-LQT controller (blue) is longer than the proposed eSLS controller (white). Indeed, when we compute the costs of the tasks for both cases, we obtained a cost of 474.3±204.3 for eSLS and 1004.7±464.2 for MPC-LQT, which also quantitatively proves the optimality of the behavior produced by our proposed controller. Indeed, after the mug is placed on the table, the recomputed controller of MPC-LQT can also put the sugar cube inside the mug successfully. However, in the optimal behavior as in eSLS, the robot decides on the location of the mug by anticipating that it will need to put the sugar cube inside, which is a missing feature in the baseline.

4.4.2 Pick-and-place task

The robot needs to grasp a cylindrical object and place it at a given position while keeping an upright orientation of the end-effector, as seen in Figure 4.1. The grasping position is defined by precise x-y positions of the center of the cylinder, whereas the z-position and the rotation around the z-axis are kept at a very low precision. It results in

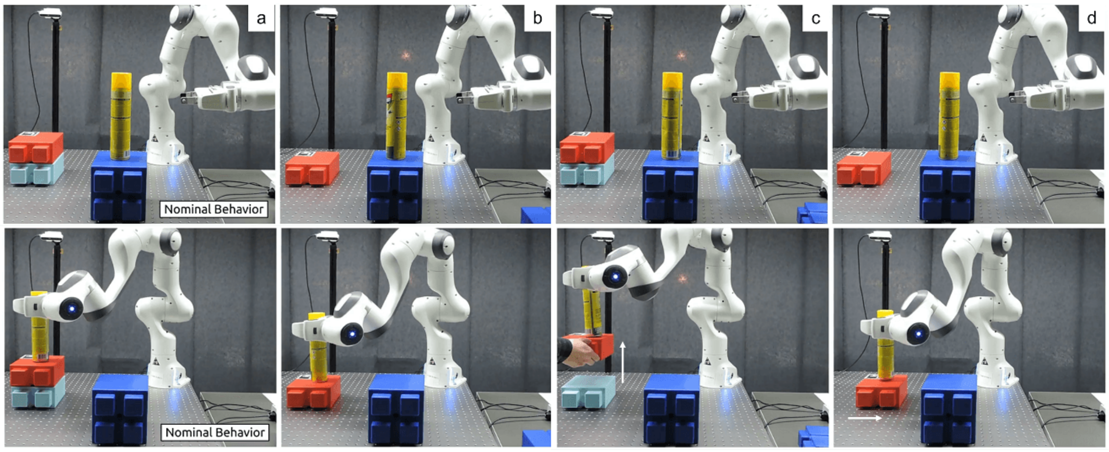


Figure 4.5: (a) Nominal behavior of the robot without any adaptation. (b) The final relative height is changed at the initial time and the robot adapts to this new height with a new plan, by remembering where it grasped the object. (c) Online adaptation by increasing the relative height of the final position after grasping the object. (d) Adaptation to change of the relative height at the beginning of the movement combined with a change in the final position introduced after the grasping phase.

different grasping heights that the robot needs to decide, by anticipating where it is going to place the object. This behaviour exploits the grasping affordances of a cylindrical object by allowing the robot to choose where and how it is going to grasp the object. The placing location is defined by precise x-y coordinates, while the z-coordinate is defined relatively to the z-coordinate when grasped. Consequently, the robot needs to remember where it grasped the object in order to place it at the relative placing z-position. Notice that without any memory feedback controller, the robot could try to place the object in wrong and dangerous locations as it can force the object to go downwards and collide with the environment.

Dynamics:

We denote $\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$, $\ddot{\boldsymbol{\theta}}$, $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, $\boldsymbol{q}$, the joint angles, the joint velocities, the joint accelerations, the end-effector positions, the end-effector velocities and the end-effector orientation in quaternion format, respectively. We define $\boldsymbol{f}_{\text{kin}}^{\text{pos}}(\cdot)$ and $\boldsymbol{f}_{\text{kin}}^{\text{orn}}(\cdot)$ as the forward kinematics functions outputting end-effector positions and orientations, respectively. We denote $\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_1)$ as the logarithmic map between the quaternions $\boldsymbol{q}_0$ and $\boldsymbol{q}_1$ and define quaternion costs with the methods described in [89].

Inspired by the work in [90], we choose to incorporate the constraints and other nonlinearities into the forward dynamics model of the state in order to alleviate the problems of calculating the Hessian of the cost function. In fact, defining the forward dynamics as such allows to have a quadratic cost function with a precision matrix that defines directly the accuracy of the state itself. This, in turn, becomes very useful when one wants to replan the motion and the controller by changing not only the joint positions, but also the end-effector positions and orientations. Joint limit constraints can be handled as a soft constraint in the cost function as this approach provides very good results already without resorting to more complicated inequality constrained optimization. However, in this work, we choose to represent these limits inside the state as well to illustrate the generalization and adaptation capability of the controller for a given quadratic cost function. We first define the joint limit violation function as $\boldsymbol{f}^{\text{lim}}(\boldsymbol{\theta})=\Big(\max{(\boldsymbol{\theta}_l-\boldsymbol{\theta},\boldsymbol{0})}+\max{(\boldsymbol{\theta}-\boldsymbol{\theta}_u,\boldsymbol{0})}\Big)^2$, where $\boldsymbol{\theta}_l$ and $\boldsymbol{\theta}_u$ are the lower and upper bounds on the joint angles, respectively. Note that when this function is nonzero, it means that the joint limits are violated, which implies that we can use this state as quadratic function to drive it to zero. We then choose to represent the state by $\boldsymbol{z}_t=[\boldsymbol{\theta}_t,\dot{\boldsymbol{\theta}}_t,\boldsymbol{x}_t,\dot{\boldsymbol{x}}_t,\text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_t),\boldsymbol{f}^{\text{lim}}]\in\mathbb{R}^{31}$ and the control by $\boldsymbol{u}_t=\ddot{\boldsymbol{\theta}}\in\mathbb{R}^7$ with the forward dynamics defined as $\Big[\boldsymbol{\theta}_{t+1} \quad \dot{\boldsymbol{\theta}}_{t+1}, \quad \boldsymbol{x}_{t+1}, \quad \dot{\boldsymbol{x}}_{t+1}, \quad \text{Log}_{\boldsymbol{q}_0}(\boldsymbol{q}_{t+1}), \quad \boldsymbol{f}^{\text{lim}}_{t+1}\Big]$ $= \Big[\boldsymbol{\theta}_t+\dot{\boldsymbol{\theta}}_t\delta t, \quad \dot{\boldsymbol{\theta}}_t+\boldsymbol{u}_t\delta t, \quad \boldsymbol{f}^{\text{pos}}_{\text{kin}}(\boldsymbol{\theta}_{t+1}), \quad \boldsymbol{J}(\boldsymbol{\theta}_{t+1})\dot{\boldsymbol{\theta}}_t, \quad \text{Log}_{\boldsymbol{q}_0}(\boldsymbol{f}^{\text{orn}}_{\text{kin}}(\boldsymbol{\theta}_{t+1})), \quad \boldsymbol{f}^{\text{lim}}(\boldsymbol{\theta}_{t+1})\Big]$.

Cost:

The cost function is designed by three key points describing the phases of *grasping*, *lifting up* and *placing* at given timesteps. For the *grasping* phase, there is high precision on the x-y axis of the end-effector, and on the end-effector velocity, whereas the precision on the z axis is left very low to give the robot the choice to grasp anywhere along the object. The *lifting* phase is implemented so as to keep the arm safe during the pick-and-place operation to avoid any obstacle on the table. Here, we have high precision only on reaching a z-position 10 cm higher than where the object was grasped during the grasping phase, and no precision on the other dimensions. For the *placing* phase, we have high precision on the x-y positions, on keeping z position the same as where it grasped the object, and on end-effector velocity. Note that this is only possible by exploiting the off-diagonal elements in the precision matrix, as explained in Section 4.3.2. For the orientations, starting from the grasping phase, we set a high precision on keeping an upright orientation (but free to turn around z-axis) in order to keep the balance of the long cylindrical object.

Results:

We implemented a 50Hz iSLS controller with a duration of 8s, which outputs the desired state and control to a 1kHz impedance controller. A vision system is implemented to track the target object location for testing the adaptation aspect of the controller to the changes in the task. We conducted several experiments with the robot to test the adaptation, reactivity and memory capabilities of the controller. We first tested the adaptation to different initial configurations. We selected randomly 3 configurations that are close to the nominal one, but still corresponding to visibly different end-effector positions and orientations, as seen in Figure 4.3. We noticed that the robot could adapt its grasping position to different z-dimensions because of the low precision on that axis. Here, the robot is aware of the possible grasping affordances of the object via our controller and exploits these to decide autonomously from which part of the object to grasp. It then remembers the grasping location and use it to place the object on the desired position without colliding with the environment.

For testing the reactivity to perturbations of the proposed controller, we applied some force to the robot changing its nominal behavior, i.e. different configurations corresponding to different end-effector positions and orientations, as can be seen in Figure 4.4. We noticed that the robot reacts to the perturbations by mostly changing its grasping locations, whilst still trying to reach the desired x-y location of the object. Even if it decides only where to grasp the object to cope with the perturbations, it can remember these locations to place the object correctly without collision as can be seen from the robot trajectories in the plot on the far right of Figure 4.4. In one example shown in the bottom-right of Figure 4.4, we changed the orientation by turning it around the z-axis. The robot's reaction to this perturbation was considerably smaller than the other perturbation types. This is indeed the expected behavior of the robot, since the error on the orientation with this change stayed very close to zero as a result of the quaternion cost function design.

Finally, we tested the proposed fast adaptation capability of the controller with memory by conducting three different experiments shown by (b), (c) and (d) in Figure 4.5. In all cases, we use the same optimized controller which produce the nominal behavior in (a), and reused it to replan in the other cases where the target locations are changed either at the beginning of the movement or on-the-fly. The change in the target location is tracked by the vision system following the marker on the object. In (b), we changed the final height of the placing location to a lower value at the initial time. In (c) and (d), the placing locations are changed on-the-fly by increasing the final height and changing the final x-y position, respectively. We recomputed online the feedforward part of the iSLS controller, namely $\boldsymbol{k}$ as described in Section 4.3.2, each time there is a significant difference between the current and detected target locations. In all cases, we see that the robot is able adapt successfully by replanning fast to new desired states without any collision, as can also be seen in the video.

4.4.3 Bimanual handover task

This task can be seen as an extension of the previous pick-and-place task using two simulated Franka Emika robots instead of one and involving a handover phase. The states $\boldsymbol{z}_t \in \mathbb{R}^{62}$ and the control commands $\boldsymbol{u}_t \in \mathbb{R}^{14}$ have the same elements as before for each of the robot, doubling the dimensionality of each.

The cost function is designed by three key points describing the phases of grasping, lifting up/handover and placing, respectively. The grasping and lifting up phases are the same as the first task for the first arm. In the handover phase, we add high precision forcing x-y positions of the end-effectors of both arms to be equal without specifying at which point, and z-position of the second arm 5 cm lower than the first arm, with zero velocities for both arms. The placing condition is the same as before, but this time for the second arm. Here, depending on where the first arm grasped the object, both arms need to coordinate to decide and plan where to meet for the handover phase. The second arm then needs to remember this information to place the object accordingly to the floor.

The video accompanying this work shows experimental results testing for different initial configurations, perturbations and replanning with respect to different final locations to place the object. For conciseness, we only show here three different behaviors produced by the proposed controller as can be seen in Figure 4.6, where each behavior is denoted with (a), (b) and (c) letters. Here, we show three phases of the movement, pick, handover and place to show the adaptation in different phases. The grasping locations are highlighted by a star sign placed at the same exact locations of the objects to distinguish between them. Before the *pick* phase, in (a) and (c), the robot is perturbed upwards and downwards respectively, while (b) is not perturbed. As expected, the robot chooses different grasping locations for each case considering the grasping affordance of the object. After the *pick* phase, and before *handover*, we introduce a new target location to place the object, which are denoted by transparent yellow cylinder, while the red ones being the original targets. In the *handover*, we can already see that the second arm can adapt its end-effector according to the grasping location of the first arm thanks to the memory feedback on the z-axis. Finally, in the *place* phase, the second robot can successfully adapt to the new target location thanks to the proposed online adaptation scheme. It could also correctly place the object without hitting the floor by remembering where it was grasped by the first arm. We argue that the proposed controller can generate smart behaviors without requiring any vision system to track the grasping locations.

4.5 Conclusion

We presented an approach for synthesizing reactive and anticipatory controllers that can remember all previous states in the horizon, which is relevant for robotic tasks requiring to have a memory of previous movements. In this context, we proposed to extend SLS controllers to have a feedforward term that can handle viapoint tasks and a Newton optimization method for solving SLS for nonlinear systems and nonquadratic cost functions. We showed that our proposed method outperforms the baseline solutions for producing optimal anticipatory behaviors that require a memory feedback. We showcased our method on high dimensional robotic systems in the presence of perturbations, and demonstrated a step towards adaptation when there is a change of the desired states in the task.

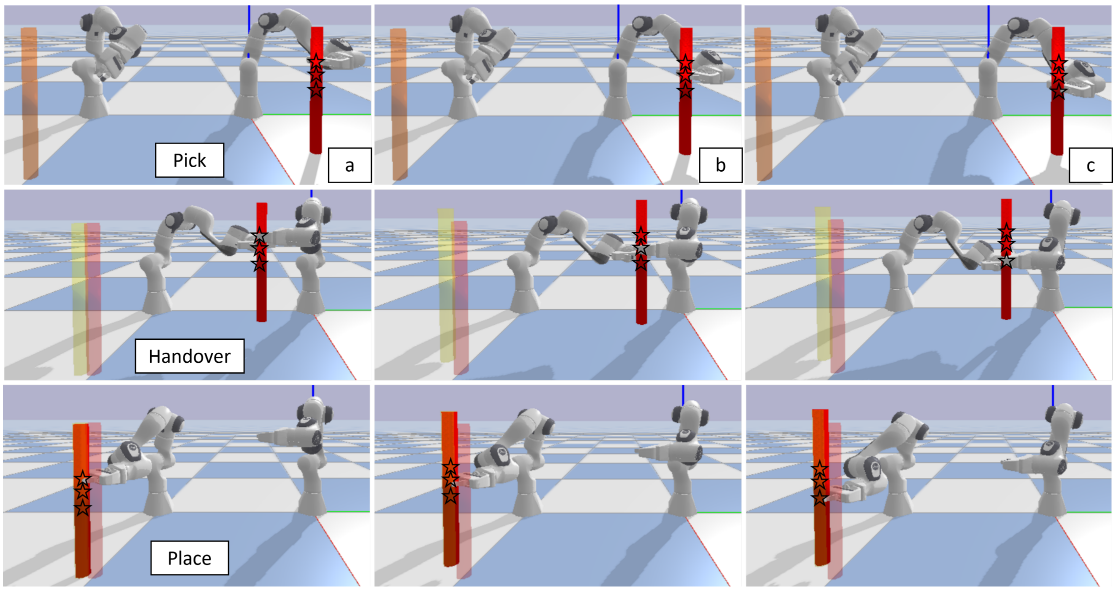


Figure 4.6: Here, we show three phases of the movement, pick, handover and place to show three different adapted behaviors, denoted as (a), (b) and (c) for each phase. The grasping locations are highlighted by a star sign placed at the same exact locations of the objects to differentiate them. Before the *pick* phase, in (a) and (c), the robot is perturbed upwards and downwards respectively, while (b) is not perturbed. After the *pick* phase, and before *handover*, a new target location to place the object which are denoted by transparent yellow cylinder is introduced, with the red ones being the original targets. Finally, in the *place* phase, the second robot can successfully adapt to new target location thanks to the proposed online adaptation scheme. It also could correctly place the object without hitting the floor by remembering where it was grasped by the first arm.

SLS offers the advantage that it does not require the experimenter to engineer the problem for each specific use case by augmenting the state-space with the relevant part of the history. In that sense, SLS is generic and formalizes this problem, since it does not require a change in the framework, by allowing the system to freely change which past states are correlated and the way they are correlated. In this work, we tackled problems where we see important practical utility of this feature. However correlating only a couple of timesteps in this work, one could also design these tasks to remember not only one timestep but several timesteps in order to capture an important part of the movement that the robot did in the past and that it needs to remember in the future to react accordingly.

5 Nullspace Methods in Planning and Control

A collection of work in neuroscience informs us that being skillful is not related to being precise [91, 92, 93, 94]. It is instead related to the exploitation of various forms of redundancy and variations in an optimal way. Figure 5.1 illustrates the various forms of redundancy that can be exploited in robotics. In this planar example, kinematic redundancy arises when a 2D point is tracked by a 3-axis robot [95, 96, 97, 98]. This form of redundancy at the morphology level is the one that is the most exploited in robotics. Most tasks do not require a specific point to be tracked, but instead consider a region or distribution of the different options in which the tip of the robot can move while satisfying the task constraints [99]. Task redundancies at the spatial level then arises when we consider for example different ways of grasping an object as was discussed in Section 4.4.2. We show in this chapter that redundancy can also be exploited at a spatiotemporal level within path planning formulations for tasks that require only sparse definition of the cost functions.

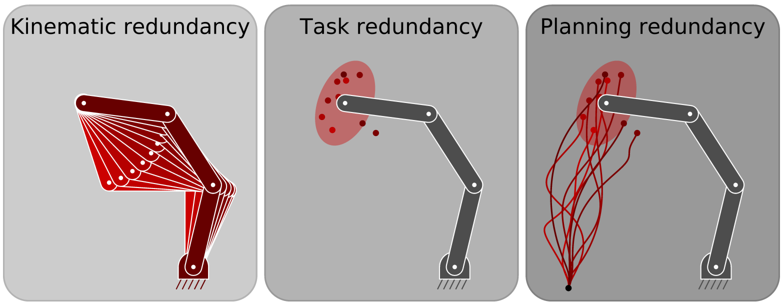


Figure 5.1: *From left to right:* kinematic/mechanical redundancy (morphology level), task redundancy (spatial level), planning redundancy (spatiotemporal level).

5.1 Inverse kinematics with nullspace structure

Redundant degrees of freedom of a robot allow robots to accomplish additional tasks in the task space at the same as achieving a primary goal. For example, a robot can reach a position and an orientation in the task-space, while still being able to move some of its joints for secondary tasks such as avoiding obstacles. In the literature, nullspace methods emerge as a solution for automatically finding the optimal way of exploiting the kinematic redundancies in this hierarchical desired task representation. Nullspace methods in optimization problems can be used as a method to solve linear-quadratic problems. As a starting basis, we give the optimization problem

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \|\boldsymbol{A}_2\boldsymbol{q} - \boldsymbol{b}_2\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{A}_1\boldsymbol{q} - \boldsymbol{b}_1 = 0, \end{aligned} \tag{5.1}$$

that admits the analytical solution

$$\boldsymbol{q}^* = \boldsymbol{A}_1^\dagger\boldsymbol{b}_1 + (\boldsymbol{A}_2\boldsymbol{N}_1)^\dagger(\boldsymbol{b}_2 - \boldsymbol{A}_2\boldsymbol{A}_1^\dagger\boldsymbol{b}_1), \tag{5.2}$$

where $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are non-invertible full row rank matrices, the pseudoinverse and nullspace matrices $\boldsymbol{N}_i$ are defined with respect to full row rank matrices and the solution is derived in Appendix A.4.1. The interpretation of Equation (5.2) is that the task related to $\boldsymbol{A}_1$ and $\boldsymbol{b}_1$ has to be fulfilled and if possible, the task related to $\boldsymbol{A}_2$ and $\boldsymbol{b}_2$ should be minimized as a secondary task.

An example of this is *hierarchical inverse kinematics*, where the problem is defined as determining a joint configuration that achieves a secondary task $\boldsymbol{f}_2(\boldsymbol{q}) = \boldsymbol{x}_2$ only if the primary task $\boldsymbol{f}_1(\boldsymbol{q}) = \boldsymbol{x}_1$ is satisfied. This problem can be written as

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \|\boldsymbol{f}_2(\boldsymbol{q}) - \boldsymbol{x}_2\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{f}_1(\boldsymbol{q}) - \boldsymbol{x}_1 = 0, \end{aligned} \tag{5.3}$$

which can be solved using a constrained version of a Gauss-Newton method [100] that sequentially linearizes the constraint and quadratizes the objective function around $\boldsymbol{q}_k$ (i.e., by approximating the Hessian of $\|\boldsymbol{f}(\boldsymbol{q})\|_2^2$ with $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^\top(\boldsymbol{q})$). The resulting linear-quadratic subproblem is

$$\begin{aligned} \min_{\delta\boldsymbol{q}} \quad & \|\boldsymbol{J}_2(\boldsymbol{q}_k)\delta\boldsymbol{q} - \delta\boldsymbol{x}_2\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{J}_1(\boldsymbol{q}_k)\delta\boldsymbol{q} - \delta\boldsymbol{x}_1 = 0, \end{aligned} \tag{5.4}$$

where $\delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_k$ and $\delta\boldsymbol{x}_i = \boldsymbol{f}_i(\boldsymbol{q}_k) - \boldsymbol{x}_i$. Using Equation (5.2), this problem can be solved as

$$\delta\boldsymbol{q}^* = \boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger\delta\boldsymbol{x}_1 + (\boldsymbol{J}_2(\boldsymbol{q}_k)\boldsymbol{N}_1(\boldsymbol{q}_k))^\dagger(\delta\boldsymbol{x}_2 - \boldsymbol{J}_2(\boldsymbol{q}_k)\boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger\delta\boldsymbol{x}_1) \tag{5.5}$$

which is the task-priority formulation of inverse kinematics described in [101]. This can even be extended to have K tasks, starting from $\delta\boldsymbol{q}_1 = \boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger\delta\boldsymbol{x}_1$ with the recursive formulation [102]

$$\delta \boldsymbol{q}_{i+1} = \delta \boldsymbol{q}_i + (\boldsymbol{J}_i \boldsymbol{N}_{i-1}^A)^\dagger (\delta \boldsymbol{x}_i - \boldsymbol{J}_i \delta \boldsymbol{q}_i), \tag{5.6}$$

where $\boldsymbol{N}_i^A = \boldsymbol{I} - \boldsymbol{J}_i^{A\dagger} \boldsymbol{J}_i^A$ is the projection matrix onto the nullspace of the augmented Jacobian matrix $\boldsymbol{J}_i^A = [\boldsymbol{J}_1^\top \boldsymbol{J}_2^\top \dots \boldsymbol{J}_i^\top]^\top$. $\boldsymbol{N}_i^A$ can also be formulated recursively as

$$\boldsymbol{N}_i^A = \boldsymbol{N}_{i-1}^A - (\boldsymbol{J}_i \boldsymbol{N}_{i-1}^A)^\dagger (\boldsymbol{J}_i \boldsymbol{N}_{i-1}^A),$$

with $\boldsymbol{N}_0^A = \boldsymbol{I}$, see [103] for details.

After finding a search direction $\delta \boldsymbol{q}_k$ using the nullspace method, one needs to perform a line search along this direction. For this, we need to use a merit function to measure progress in both the objective function and the constraints. A standard one is the quadratic penalty merit function with $M(\boldsymbol{q}) = \|\boldsymbol{f}_2(\boldsymbol{q}) - \boldsymbol{x}_2\|_2^2 + \rho \|\boldsymbol{f}_1(\boldsymbol{q}) - \boldsymbol{x}_1\|_2^2$. A step $\alpha_k \delta \boldsymbol{q}_k$ is accepted the following sufficient decrease condition holds:

$$M(\boldsymbol{q}_k + \alpha_k \delta \boldsymbol{q}_k) \leq M(\boldsymbol{q}_k) \tag{5.7}$$

where ρ is chosen large enough [104]. We follow the heuristic update rule in augmented Lagrangian method presented also in Chapter 3.

The pseudoinverse $\boldsymbol{J}^\dagger$ can be computed in different ways according to the rank of $\boldsymbol{J}$. If $\boldsymbol{J}$ is full row-rank, then $\boldsymbol{J}^\dagger = \boldsymbol{J}^\top (\boldsymbol{J}\boldsymbol{J}^\top)^{-1}$, or else if $\boldsymbol{J}$ is full column-rank, then $\boldsymbol{J}^\dagger = (\boldsymbol{J}^\top \boldsymbol{J})^{-1} \boldsymbol{J}^\top$. Using a singular value decomposition (SVD) to compute the pseudoinverse provides a more general method as it can be used either in the rank-deficient case or in the full-rank case. For simplicity, we will adopt in this chapter the SVD perspective for the computation of pseudoinverses. Pseudoinverse computation details are given in Appendix A.2.1.

In inverse kinematics, we can encounter kinematic singularities arising from the singularity of the Jacobian matrix which can result in high velocities that could potentially damage the robot. These singularities can be handled using a damped (or regularized) pseudoinverse computed as

$$\boldsymbol{J}^\ddagger = \boldsymbol{J}^\top (\boldsymbol{J}\boldsymbol{J}^\top + \lambda \boldsymbol{I}_{\boldsymbol{x}})^{-1} = (\boldsymbol{J}^\top \boldsymbol{J} + \lambda \boldsymbol{I}_{\boldsymbol{u}})^{-1} \boldsymbol{J}^\top, \tag{5.8}$$

where λ is the damping factor. $\boldsymbol{I}_{\boldsymbol{x}}$ and $\boldsymbol{I}_{\boldsymbol{u}}$ are identity matrices with dimensions of $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively. Using SVD, we can show the existence of the nullspace in the presence of damping (see Appendix A.2.1). Note that the nullspace formulations can contain kinematic singularities because of the computation of $(\boldsymbol{J}_2 \boldsymbol{N}_1)^\dagger$ in the same way as using damped pseudoinverses [105].

5.2 Nullspace structure in linear quadratic tracking (LQT)

Nullspace structure in MPC allows us to exploit the redundancy in space-time. At each time step, we can have tasks that require different precisions and that are redundant in space dimensions, with different priorities. In the same way, we can have tasks that are redundant in time dimensions for each space dimension.

When we consider the batch version of LQT (see Section 2.2.2), the purpose of the cost on the control commands is to prevent high, non-smooth control actions. Mathematically, it allows us to regularize and to invert $\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}}$ (by adding a full-rank $\boldsymbol{R}$) which may be otherwise non-invertible due to the sparse definition of the cost on the state (e.g. a final timestep cost only). There might be infinitely many choice of $\boldsymbol{R}$ that would satisfy our problem requirements, but setting a specific one, we constrain the problem to have a unique solution. We can see this by considering the problem in Equation (2.7) without the cost on the control commands as

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \|\boldsymbol{x}-\boldsymbol{\mu}_{\boldsymbol{x}}\|_{\boldsymbol{Q}}^2 \\ \text{s.t.} \quad & \boldsymbol{x}=\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0+\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}. \end{aligned} \tag{5.9}$$

We can transform this constrained problem into an unconstrained problem as

$$\min_{\boldsymbol{u}} \quad \|\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0+\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}-\boldsymbol{\mu}_{\boldsymbol{x}}\|_{\boldsymbol{Q}}^2,$$

which can be solved by taking the derivative of the cost function with respect to $\boldsymbol{u}$ and equating it to zero as

$$\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}=\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}(\boldsymbol{\mu}_{\boldsymbol{x}}-\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0)$$

This equation suggests that we can exploit the nullspace of $\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}}$ as

$$\hat{\boldsymbol{u}}=(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}})^{\dagger}\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}(\boldsymbol{\mu}_{\boldsymbol{x}}-\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0)+\boldsymbol{N}\boldsymbol{y},$$

where $\boldsymbol{N}$ is the nullspace of $\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}}$ and $\boldsymbol{y}$ is an arbitrary vector. Note that this formulation is only an example of what we can set as the nullspace and was used in [106]. We show now how to derive the nullspace structure in a proper way by defining mathematically the term *hierarchical tasks.*

Definition 1. *Two tasks defined by the stacked reference states* $\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}$*,* $\boldsymbol{\mu}_{\boldsymbol{x}}^{(2)}$ *and block-diagonal precision matrices* $\boldsymbol{Q}^{(1)}$*,* $\boldsymbol{Q}^{(2)}$ *are said to be prioritized one over another (*$1>2$*) if the controller achieves the same cost values* $\|\boldsymbol{x}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|_{\boldsymbol{Q}^{(1)}}^2$ *regardless of whether there is a secondary task.*

Let $\boldsymbol{x}^{(1)},\boldsymbol{u}^{(1)}=\arg\min_{\boldsymbol{x},\boldsymbol{u}} c_1(\boldsymbol{x},\boldsymbol{u})$ s.t. $\boldsymbol{x}=\boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0+\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}$ where $c_1(\boldsymbol{x},\boldsymbol{u})=\|\boldsymbol{x}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|_{\boldsymbol{Q}^{(1)}}^2+\|\boldsymbol{u}\|_{\boldsymbol{R}_{d1}}^2$ is the cost function to solve for a primary task defined by the superscript (1). We would like to find out the conditions to hierarchically add some arbitrary pair $\{\boldsymbol{\delta x},\boldsymbol{\delta u}\}$ to the primary task solution pair $\{\boldsymbol{x}^{(1)},\boldsymbol{u}^{(1)}\}$, i.e., without changing the value of the primary objective $c_1^x(\boldsymbol{x}^{(1)})=\|\boldsymbol{x}^{(1)}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|^2_{\boldsymbol{Q}^{(1)}}$. The new objective function can be written as

$$\begin{aligned} c_{1\delta}^x(\boldsymbol{\delta x}) &= \|\boldsymbol{x}^{(1)}+\boldsymbol{\delta x}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|^2_{\boldsymbol{Q}^{(1)}}, \\ &= \underbrace{\|\boldsymbol{x}^{(1)}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|^2_{\boldsymbol{Q}^{(1)}}}_{c_1^x(\boldsymbol{x}^{(1)})} + \underbrace{\|\boldsymbol{\delta x}\|^2_{\boldsymbol{Q}^{(1)}} + 2(\boldsymbol{x}^{(1)}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)})^\top \boldsymbol{Q}^{(1)}\boldsymbol{\delta x}}_{c_\delta(\boldsymbol{\delta x})}, \end{aligned}$$

Here, we can solve $c_\delta(\boldsymbol{\delta x})=0$ to find the solution set of $\boldsymbol{\delta x}$ that does not affect the primary cost. However, $\boldsymbol{\delta x}$ is constrained to the system dynamics by $\boldsymbol{\delta x}=\boldsymbol{S_u}\boldsymbol{\delta u}$. We can therefore replace it by this expression to obtain

$$c_\delta(\boldsymbol{\delta u}) = \|\boldsymbol{S_u}\boldsymbol{\delta u}\|^2_{\boldsymbol{Q}^{(1)}} + 2(\boldsymbol{x}^{(1)}-\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)})^\top \boldsymbol{Q}^{(1)}\boldsymbol{S_u}\boldsymbol{\delta u} = 0, \tag{5.10}$$

which admits the set of solutions $\{\boldsymbol{0}, 2(\boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}-\boldsymbol{x}^{(1)}), \boldsymbol{N}_{\boldsymbol{S_u}}\boldsymbol{y}, \boldsymbol{N}_{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}\boldsymbol{y}\}$, where $\boldsymbol{y}$ is arbitrary. If we remove the first two trivial particular solutions from the set, and remark that $\boldsymbol{\delta x}=\boldsymbol{S_u}\boldsymbol{\delta u}=\boldsymbol{S_u}\boldsymbol{N}_{\boldsymbol{S_u}}\boldsymbol{y}=\boldsymbol{0}$, the only non-trivial solution left is $\boldsymbol{\delta u}=\boldsymbol{N}_{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}\boldsymbol{y}$. This means that any arbitrary $\boldsymbol{y}$ that we choose will not alter the task success defined as the cost value on the state. Note that this nullspace seems different from $\boldsymbol{N}_{\boldsymbol{S_u}^\top\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}$, which is the one presented in [106], however, one can show that they represent the same nullspaces as the nullspace of $\boldsymbol{A}$ and $\boldsymbol{A}^\top\boldsymbol{A}$ are the same, i.e., $\boldsymbol{N}_{\boldsymbol{S_u}^\top\boldsymbol{Q}^{(1)}\boldsymbol{S_u}} = \boldsymbol{N}_{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}$.

Our goal is to minimize a secondary objective function such that it achieves hierarchically the first task. Denoting $\boldsymbol{N}=\boldsymbol{N}_{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}$, the solution $\boldsymbol{\delta u}$ that achieves this objective can be written as

$$\begin{aligned} \boldsymbol{\delta u}^* &= \boldsymbol{N}(\mathop{\mathsf{arg\,min}}_{\boldsymbol{y}} \|\boldsymbol{S_x}\boldsymbol{x}_0 + \boldsymbol{S_u}(\boldsymbol{u}^{(1)}+\boldsymbol{N}\boldsymbol{y}) - \boldsymbol{\mu}_{\boldsymbol{x}}^{(2)}\|^2_{\boldsymbol{Q}^{(2)}} + \|\boldsymbol{u}^{(1)}+\boldsymbol{N}\boldsymbol{y}\|^2_{\boldsymbol{R}}), && (5.11) \\ &= \boldsymbol{N}\Big(\boldsymbol{N}(\boldsymbol{S_u}^\top\boldsymbol{Q}^{(2)}\boldsymbol{S_u}+\boldsymbol{R})\boldsymbol{N}\Big)^\dagger \boldsymbol{N}\Big(\boldsymbol{S_u}^\top\boldsymbol{Q}^{(2)}(\boldsymbol{\mu}_{\boldsymbol{x}}^{(2)}-\boldsymbol{x}^{(1)}) - \boldsymbol{R}\boldsymbol{u}^{(1)}\Big), && (5.12) \\ &= \Big(\boldsymbol{N}(\boldsymbol{S_u}^\top\boldsymbol{Q}^{(2)}\boldsymbol{S_u}+\boldsymbol{R})\boldsymbol{N}\Big)^\dagger \Big(\boldsymbol{S_u}^\top\boldsymbol{Q}^{(2)}(\boldsymbol{\mu}_{\boldsymbol{x}}^{(2)}-\boldsymbol{x}^{(1)}) - \boldsymbol{R}\boldsymbol{u}^{(1)}\Big), && (5.13) \end{aligned}$$

where the second and third equalities are derived using the first and the third properties listed in Appendix A.2.2, respectively.

5.3 Experiments

5.3.1 Proof-of-concept examples

Our first example consists in reaching a goal position *Goal*, while passing through viapoints with different precisions and different hierarchies, see Figure 5.2. The primary viapoint represented by V_1^t is to be on the line (depicted by the thin ellipsoid) at time step $t = T/2$ (used as half-time of the execution for the plot). This task creates redundancy in space-time because according to the upcoming secondary tasks and previous actions taken, the position on the line at time step t can change. The secondary viapoint represented by V_2^t has an isotropic precision with no redundancy in any space dimension, meaning that at time step t, the objective of the secondary task is to be at the center of the pink circle. Figure 5.2(a) shows an LQR execution with a unit mass double integrator starting from the initial position (shown by the cross), with 2 viapoint tasks at time step t with hierarchies $V_1^t > V_2^t$ and the final task *Goal*. At time step t, the algorithm finds the best compromise between accomplishing both tasks, and taking into account their priorities. Such a compromise can be found intuitively as the intersection point between the thin ellipse and the projection of the center of the pink circle onto the thin ellipse, hence the dashed gray line.

Figure 5.2(b) shows the trajectory of the same agent, with the addition of a tertiary task V_3^{t+1} to be achieved at time step $t+1$ and whose variance is represented by a purple circle. Since this position can be reached without disturbing the primary and secondary tasks, the trajectory changes so as to accomplish all three tasks. Figure 5.2(c) shows that the addition of another quaternary task V_4^{t+1} to be achieved at time step $t+1$, represented by a turquoise circle, does not change the trajectory shown in Figure 5.2(b) because this task is in conflict with the tertiary task and is thus neglected due to its priority.

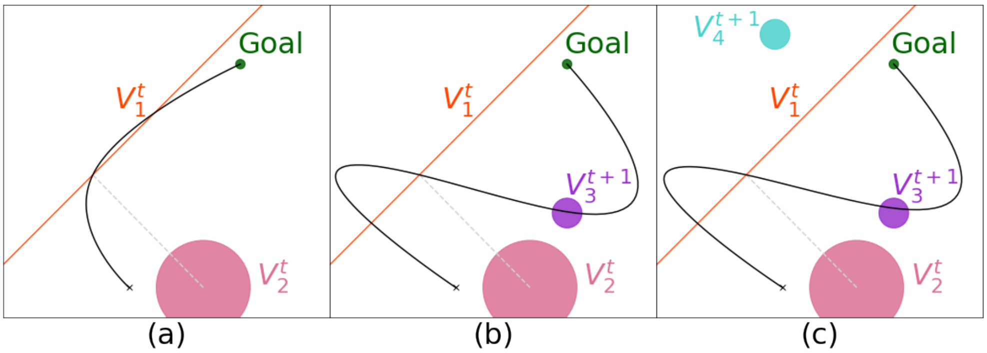


Figure 5.2: LQR with initially at the position shown by the cross and (a) 2 intermediary tasks at time step t with hierarchies $V_1^t > V_2^t$ and the final task *Goal*, (b) 2 intermediary tasks at time step t, 1 intermediary task at time step t+1 with hierarchies $V_1^t > V_2^t > V_3^{t+1}$, and the final task *Goal*, (c) 2 intermediary tasks at time step t, 2 intermediary tasks at time step $t+1$ with hierarchies $V_1^t > V_2^t > V_3^{t+1} > V_4^{t+1}$, and the final task *Goal*. Tasks are represented with Gaussian ellipsoids, while the resulting trajectory is represented by black lines. The grey dashed line represents the shortest line between the mean of the Gaussian ellipsoid of V_2^t and the line ellipsoid (thin Gaussian ellipsoid) V_1^t.

Figure 5.3 shows an example where the nullspace formulation has an advantage over the naive attempt of minimizing both tasks errors at the same, using an hyperparameter δ to set importance weights between the tasks, where $\delta = 0$ means that only the primary task is achieved and for a large δ only the secondary task is achieved. In this example, we want to reach the point G_T at the final time step T with a given variance represented by

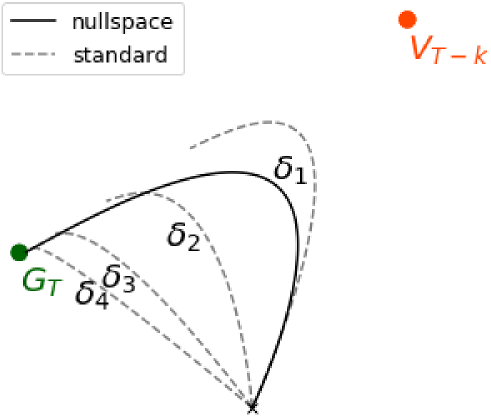


Figure 5.3: Comparison between nullspace (solid black line) and standard formulation (dashed lines) with scaling δ_1, δ_2, δ_3 and δ_4 are 1, 0.1, 0.01 and 0.001 respectively, using a point mass object with double integrator dynamics. The agent has to go to the goal point G_T as its primary task, at the final time step T, with variance shown by a green circle. If possible, it should pass through a viapoint V_{T-k}, as a secondary task, at the time step $T-k$, with the same variance shown by orange circle. When the secondary task is far away from the primary task in space and time, nullspace planning tries to pass through V_{T-k}, and still succeed to accomplish the primary task. On the other hand, no matter the scaling of the cost, standard control cannot have the same performance.

the green circle as the primary task. We also want to pass through the viapoint V_{T-k}, k time steps before the final time step T, with a given variance represented by the orange circle as the secondary task. With real-world robots, we have restrictions on the norm of the control commands, hence achieving both tasks would become impossible if they are too far away in space dimensions but very close in time dimensions. In this Figure, any naive attempts to accomplish the tasks with a hierarchy imposed by δ fails, except for very small values, which are not interesting, because we know intuitively that we can still do much better by trying to achieve the secondary task than achieving only the primary task.

5.3.2 Robot Simulation

The setup consists of two robotic agents, represented by point mass agents shown in Figure 5.5 (black and red). They have to reach a goal position as a primary task while meeting with each other at the halfway of their movement (at t=T/2) as a secondary task. One can think of many real life applications that can be represented by this example: a bimanual robot passing an object from one hand to the other and then using both hands individually, or two mobile robots that have to exchange some products in a factory before moving to their respective location. These applications require the robots to have multiple layers of workload, such as safety, main mission completion, social navigation, etc. Considering that these two agents are perturbed during their execution to avoid some obstacles, we expect them to perform their main mission and do the secondary tasks in the nullspace. In Figure 5.5, we see that any perturbation that is not in conflict between the main task of reaching the goal can be performed using the nullspace structure by autonomously modifying the meeting position.

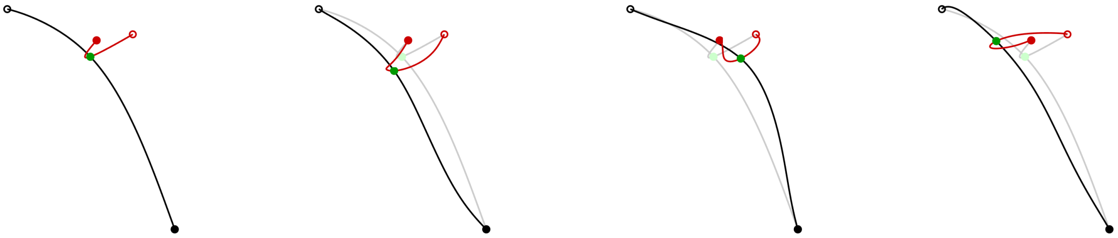


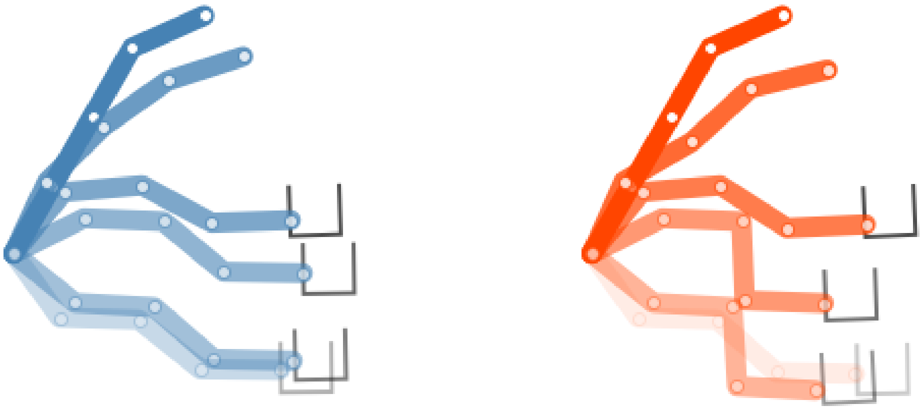
Figure 5.4: Two agents (black and red) have a primary task of reaching their goal positions at time step T and a secondary goal to meet at some position at time step $T/2$. If obstacles perturb their executions, we observe the nullspace effect with a shift of the meeting position.

We can think of another robotic application where we can exploit the redundancies in space-time dimensions of the robot. In Fig. 5.5, we have a 4 DoF robot controlled by acceleration commands over T time steps. The 8-dimensional state consists of 4-dimensional joint positions and 4-dimensional joint velocities. Therefore we have a total of $8T$ DoF along the trajectory of the robot. The robot has to pick up a cup at time step $T/2$ and place it at another location at time step T. We assume that picking up the cup and placing it both spend 8 DoF, which makes a total of 16 DoF used. The resulting trajectory is shown in Figure 5.5(a). Then, we impose a secondary objective to hold a 90 degree angle with its last 2 joints after picking up the cup. This is helpful for the robot to hold the cup with a better manipulability, as a human would do. Note that the secondary objective uses only 2 DoF at each time step, with a total of $2 \times (T/2)$ DoF used. The resulting trajectory of Figure 5.5*(b)* shows that the robot is able to keep 90 degree angles only when it is not in conflict with the tasks of picking up the cup and placing it.

5.4 Nullspace feedback controller in system level synthesis (SLS)

In this section, we would like to find a feedback controller as in LQT to achieve hierarchical tasks in optimal control. To find such a controller, we describe the problem in terms of the system level synthesis (SLS) framework Chapter 4 as it allows us to work with closed-loop maps instead of open-loop trajectories.

To find a *nullspace feedback controller* to achieve a secondary task without disturbing the



(a) Without nullspace (b) With nullspace

Figure 5.5: A robotic application of nullspace structure within LQR.

first one, we can perform sensitivity analysis on the cost function for the primary task whose solution is given by $\{\boldsymbol{\Phi}_x^{(1)}, \boldsymbol{\Phi}_u^{(1)}, \boldsymbol{d}_{\boldsymbol{x}}^{(1)}, \boldsymbol{d}_{\boldsymbol{u}}^{(1)}\}$. Note that the primary task solution is found by minimizing $c_1(\boldsymbol{x}, \boldsymbol{u}){=}\|\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}\|^2_{\boldsymbol{Q}^{(1)}} + \|\boldsymbol{u}\|^2_{\boldsymbol{R}_{d1}}$ as in the previous section for LQT, but using SLS method presented in Chapter 4. Following the same procedure, we then add $\{\boldsymbol{\delta\Phi_x}, \boldsymbol{\delta\Phi_u}, \boldsymbol{\delta d_x}, \boldsymbol{\delta d_u}\}$ to the solution such that the primary cost on the task level stays the same, but it accomplishes at the same time a secondary task.

We take (5.10), replace $\boldsymbol{\delta u}$ by $\boldsymbol{\delta\Phi_u w} + \boldsymbol{\delta d_u}$ and $\boldsymbol{x}^{(1)}$ by $\boldsymbol{\Phi}_x^{(1)}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{x}}^{(1)}$ to obtain

$$\begin{aligned}
c_\delta(\boldsymbol{\delta\Phi_u}, \boldsymbol{\delta d_u}) &= \mathbb{E}_{\boldsymbol{w}}[\|\boldsymbol{S_u}(\boldsymbol{\delta\Phi_u w} + \boldsymbol{\delta d_u})\|^2_{\boldsymbol{Q}^{(1)}} + \\
&\quad 2(\boldsymbol{\Phi}_x^{(1)}\boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{x}}^{(1)} - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)})^\top \boldsymbol{Q}^{(1)}\boldsymbol{S_u}(\boldsymbol{\delta\Phi_u w} + \boldsymbol{\delta d_u})], \\
&= \underbrace{\|\boldsymbol{S_u}\boldsymbol{\delta\Phi_u}\boldsymbol{\Sigma}_w^{\frac{1}{2}}\|^2_{\boldsymbol{Q}^{(1)}} + 2\,\mathrm{Tr}[\boldsymbol{\Phi}_x^{(1)}\Sigma\boldsymbol{Q}^{(1)}\boldsymbol{S_u}\boldsymbol{\delta\Phi_u}]}_{J_\delta(\boldsymbol{\delta\Phi_u})} + \\
&\quad \underbrace{\|\boldsymbol{\delta d_u}\|^2_{\boldsymbol{Q}^{(1)}} + (\boldsymbol{d}_{\boldsymbol{x}}^{(1)} - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)})^\top \boldsymbol{Q}^{(1)}\boldsymbol{S_u}\boldsymbol{\delta d_u},}_{J_\delta(\boldsymbol{\delta d_u})} \\
&= J_\delta(\boldsymbol{\delta\Phi_u}) + J_\delta(\boldsymbol{\delta d_u}),
\end{aligned} \tag{5.14}$$

whose non-trivial set of solutions for $\{\boldsymbol{\delta d_u}, \boldsymbol{\delta\Phi}_{\boldsymbol{u}}^i\}$ lies in the nullspace of $\{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}, \boldsymbol{Q}^{(1,i:)}\boldsymbol{S}_{\boldsymbol{u}}^{i:}\}$ denoted as $\{\boldsymbol{N}{=}\boldsymbol{N}_{\boldsymbol{Q}^{(1)}\boldsymbol{S_u}}, \boldsymbol{N}^i{=}\boldsymbol{N}_{\boldsymbol{Q}^{(1,i:)}\boldsymbol{S}_{\boldsymbol{u}}^{i:}}\}$, i.e., $\boldsymbol{\delta d_u}{=}\boldsymbol{N y}$ and $\boldsymbol{\delta\Phi}_{\boldsymbol{u}}^i{=}\boldsymbol{N}^i\boldsymbol{Y}^i$, where $\boldsymbol{y}$ and $\boldsymbol{Y}^i$ are arbitrary. Note that the index i represents the block column index for the corresponding matrix. For example, $\boldsymbol{Y}^i$ corresponds to the i^{th} block column of $\boldsymbol{Y}$, and $\boldsymbol{Y}^{i:}$ corresponds to the matrix that is obtained by extracting all the elements of $\boldsymbol{Y}$ after (and including) the i^{th} block column. Then the optimization problem that we are trying to solve becomes

$$\begin{aligned}\min_{\boldsymbol{\delta\Phi_u},\boldsymbol{\delta d_u}} \quad & \|\boldsymbol{\Phi_x w}+\boldsymbol{d_x}-\boldsymbol{\mu_x}^{(2)}\|^2_{\boldsymbol{Q}^{(2)}} + \|\boldsymbol{\Phi_u w}+\boldsymbol{d_u}-\boldsymbol{u}^{(2)}\|^2_{\boldsymbol{R}} \\ \text{s.t.} \quad & \boldsymbol{\Phi_x} = \boldsymbol{S}_x + \boldsymbol{S}_u\boldsymbol{\Phi_u}, \\ & \boldsymbol{d_x} = \boldsymbol{S_u d_u}, \\ & \boldsymbol{\Phi_u} = \boldsymbol{\Phi}_u^{(1)} + \boldsymbol{\delta\Phi_u}, \\ & \boldsymbol{d_u} = \boldsymbol{d_u}^{(1)} + \boldsymbol{\delta d_u}, \\ & \boldsymbol{\delta\Phi^i_u} = \boldsymbol{N}^i\boldsymbol{Y}^i, \\ & \boldsymbol{d_u} = \boldsymbol{Ny}, \\ & \boldsymbol{\delta\Phi_u} \in \mathcal{L}\end{aligned} \tag{5.15}$$

whose objective can be decomposed as two parts (as in Appendix A.3), $J_2(\boldsymbol{\Phi_u}, \boldsymbol{d_u}){=}J_2(\boldsymbol{d_u}){+}\sum_i J_2^i(\boldsymbol{\Phi^i_u})$, where $J_2(\boldsymbol{d_u}) = \|\boldsymbol{S_u d_u} - \boldsymbol{\mu_x}^{(2)}\|^2_{\boldsymbol{Q}^{(2)}} + \|\boldsymbol{d_u} - \boldsymbol{\mu_u}^{(2)}\|^2_{\boldsymbol{R}}$ and $J_2^i(\boldsymbol{\Phi^i_u}) = \|(\boldsymbol{S^i_x} + \boldsymbol{S_u^{i:}}\boldsymbol{\Phi^i_u})\boldsymbol{\sigma}^i_w\|^2_{\boldsymbol{Q}^{(2,i:)}} + \|\boldsymbol{\Phi^i_u}\boldsymbol{\sigma}^i_w\|^2_{\boldsymbol{R}}$ such that it achieves hierarchically the first task, i.e, $\boldsymbol{\Phi^i_u} = \boldsymbol{\Phi}_u^{(1,i)} + \boldsymbol{N}^i\boldsymbol{Y}^i$ and $\boldsymbol{d_u} = \boldsymbol{d_u}^{(1)} + \boldsymbol{Ny}$. Inserting these constraints inside, we obtain

$$\begin{aligned} J_2(\boldsymbol{y}) &= \|\boldsymbol{d_x}^{(1)} + \boldsymbol{S_u N y} - \boldsymbol{\mu_x}^{(2)}\|^2_{\boldsymbol{Q}^{(2)}} + \|\boldsymbol{d_u}^{(1)} + \boldsymbol{Ny} - \boldsymbol{\mu_u}^{(2)}\|^2_{\boldsymbol{R}}, \\ J_2^i(\boldsymbol{Y}^i) &= \|(\boldsymbol{\Phi}_x^{(1,i)} + \boldsymbol{S_u^{i:}}\boldsymbol{N}^i\boldsymbol{Y}^i)\boldsymbol{\sigma}^i_w\|^2_{\boldsymbol{Q}^{(2,i:)}} + \|(\boldsymbol{\Phi}_u^{(1,i)} + \boldsymbol{N}^i\boldsymbol{Y}^i)\sigma^i_w\|^2_{\boldsymbol{R}^{i:}} \end{aligned}$$

which can be minimized analytically for $\boldsymbol{y}$ and $\boldsymbol{Y}^i$ to find

$$\begin{aligned} \boldsymbol{\delta d_u^*} &= \boldsymbol{N} \operatorname*{arg\,min}_{\boldsymbol{y}} J_2(\boldsymbol{d_u}^{(1)} + \boldsymbol{Ny}), \\ &= \Big(\boldsymbol{N}(\boldsymbol{S_u^\top Q}^{(2)}\boldsymbol{S_u})\boldsymbol{N}\Big)^\dagger \Big(\boldsymbol{S_u^\top Q}^{(2)}(\boldsymbol{\mu_x}^{(2)} - \boldsymbol{d_x}^{(1)}) + \boldsymbol{R}(\boldsymbol{\mu_u}^{(2)} - \boldsymbol{d_u}^{(1)})\Big), \end{aligned} \tag{5.16}$$

and

$$\begin{aligned} \boldsymbol{\delta\Phi_u^{i*}} &= \boldsymbol{N}^i \operatorname*{arg\,min}_{\boldsymbol{Y}^i} J_2^i(\boldsymbol{\Phi}_u^{(1,i)} + \boldsymbol{N}^i\boldsymbol{Y}^i), \\ &= -\Big(\boldsymbol{N}^i(\boldsymbol{S_u^{i:}}^\top\boldsymbol{Q}^{(2,i:)}\boldsymbol{S_u^{i:}})\boldsymbol{N}^i\Big)^\dagger \Big(\boldsymbol{S_u^{i:}}^\top\boldsymbol{Q}^{(2,i:)}\boldsymbol{\Phi}_x^{(1,i)} + \boldsymbol{R}\boldsymbol{\Phi}_u^{(1,i)}\Big) \end{aligned} \tag{5.17}$$

Note that such a feedback controller allows us to apply adaptation methods proposed in Chapter 4 into the hierarchical task definition by noticing that the desired states $\boldsymbol{\mu_x}^{(1)}$ and $\boldsymbol{\mu_x}^{(2)}$ only appear in $\boldsymbol{\delta d_u^*}$ linearly. The same observation allowed us to store some matrices and adapt very fast by a simple matrix vector multiplication in the case of a change in the desired states. We can apply the same ideas here to get an optimal solution of the same problem with different desired states. This observation is becoming more interesting when we consider the nullspace structure in iSLS in the next section.

5.5 Bilevel optimization of hierarchical optimal control

The nullspace definition and emergence of the nullspace matrices are similar to the inverse kinematics previously discussed in Section 5.1 once we establish what we mean by *hierarchical tasks*. We follow again Section 5.1 to look at the hierarchical inverse kinematics problem from the perspective of bilevel optimization.

First we notice that the solution to the problem in (5.3) is also the solution to the following problem (see Appendix A.4.2)

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \|\boldsymbol{f}_2(\boldsymbol{q}) - \boldsymbol{x}_2\|_2^2 \\ \text{s.t.} \quad & \|\boldsymbol{f}_1(\boldsymbol{q}) - \boldsymbol{x}_1\|_2^2 \quad \text{is minimum.} \end{aligned} \tag{5.18}$$

and the iterations in constrained Gauss-Newton are the same iterations in both problems. Using Definition 1, we apply this idea to define the hierarchical optimal control problem as

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \|\boldsymbol{g}_2(\boldsymbol{u})\|_2^2 + \|\boldsymbol{u}\|_{\boldsymbol{R}}^2 \\ \text{s.t.} \quad & \|\boldsymbol{g}_1(\boldsymbol{u})\|_2^2 \quad \text{is minimum.} \end{aligned} \tag{5.19}$$

where the costs $\|\boldsymbol{g}_i(\boldsymbol{u})\|_2^2$ describe the costs on the trajectory level only without including the quadratic terms on the control, such that there is some planning redundancy. We will see later how this redundancy appears in the subproblems for solving (5.19). Note that here we assume these costs include the dynamics model as was done in Section 3.3.4 in the form $\boldsymbol{x} = \boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u})$.

To solve (5.19), we first transform this problem into a single level optimization problem by replacing the constraint with its KKT conditions $\nabla \boldsymbol{g}_1(\boldsymbol{u})^\top \boldsymbol{g}_1(\boldsymbol{u}) = \boldsymbol{0}$ as follows

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \|\boldsymbol{g}_2(\boldsymbol{u})\|_2^2 + \|\boldsymbol{u}\|_{\boldsymbol{R}}^2 \\ \text{s.t.} \quad & \nabla \boldsymbol{g}_1(\boldsymbol{u})^\top \boldsymbol{g}_1(\boldsymbol{u}) = \boldsymbol{0} \end{aligned} \tag{5.20}$$

We linearize it around $\boldsymbol{u}_k$ and the subproblem of the Gauss-Newton method is

$$\begin{aligned} \min_{\Delta \boldsymbol{u}} \quad & \|\nabla \boldsymbol{g}_2(\boldsymbol{u}_k)\Delta \boldsymbol{u} + \boldsymbol{g}_2(\boldsymbol{u}_k)\|_2^2 + \|\Delta \boldsymbol{u} + \boldsymbol{u}_k\|_{\boldsymbol{R}}^2 \\ \text{s.t.} \quad & \nabla \boldsymbol{g}_1(\boldsymbol{u}_k)^\top \boldsymbol{g}_1(\boldsymbol{u}_k) + \nabla \boldsymbol{g}_1(\boldsymbol{u}_k)^\top \nabla \boldsymbol{g}_1(\boldsymbol{u}_k) \Delta \boldsymbol{u} = \boldsymbol{0} \end{aligned} \tag{5.21}$$

which is a linearly constrained quadratic program with respect to $\Delta \boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}_k$, that can be solved by the constrained Gauss-Newton method as in Appendix A.4.1. We can use the nullspace methods to re-write the equality constraint as

$$\Delta \boldsymbol{u} = -\nabla \boldsymbol{g}_1(\boldsymbol{u}_k)^\dagger \boldsymbol{g}_1(\boldsymbol{u}_k) + \boldsymbol{N}_1(\boldsymbol{u}_k)\boldsymbol{y}, \tag{5.22}$$

$$= \Delta \boldsymbol{u}^{(1)} + \boldsymbol{N}_1(\boldsymbol{u}_k)\boldsymbol{y}, \tag{5.23}$$

with an arbitrary $\boldsymbol{y}$ (see Equation (A.36)). Inserting this into Equation (5.21), we obtain

$$\min_{\boldsymbol{y}} \quad \|\nabla \boldsymbol{g}_2(\boldsymbol{u}_k)\Big(\Delta \boldsymbol{u}^{(1)} + \boldsymbol{N}_1(\boldsymbol{u}_k)\boldsymbol{y}\Big) + \boldsymbol{g}_2(\boldsymbol{u}_k)\|_2^2 + \|\Delta \boldsymbol{u}^{(1)} + \boldsymbol{N}_1(\boldsymbol{u}_k)\boldsymbol{y} + \boldsymbol{u}_k\|_{\boldsymbol{R}}^2 \tag{5.24}$$

which is in the same form as in the linear quadratic case Equation (5.11). We can then directly give the solution as

$$\boldsymbol{y}^* = -\Big(\boldsymbol{N}_1(\nabla \boldsymbol{g}_2^\top \nabla \boldsymbol{g}_2 + \boldsymbol{R})\boldsymbol{N}_1\Big)^\dagger \Big(\nabla \boldsymbol{g}_2^\top(\nabla \boldsymbol{g}_2 \Delta \boldsymbol{u}^{(1)} + \boldsymbol{g}_2) + \boldsymbol{R}(\Delta \boldsymbol{u}^{(1)} + \boldsymbol{u}_k)\Big) \tag{5.25}$$

which gives the optimal solution to the subproblem of hierarchical iLQR as

$$\Delta \boldsymbol{u}^* = \Delta \boldsymbol{u}^{(1)} + \boldsymbol{N}_1(\boldsymbol{u}_k)\boldsymbol{y}^*. \tag{5.26}$$

As usual, we choose a merit function for the line search on the solution $\Delta \boldsymbol{u}^*$ as $M(\boldsymbol{q}) = \|\boldsymbol{g}_2(\boldsymbol{u})\|_2^2 + \|\boldsymbol{u}\|_{\boldsymbol{R}}^2 + \mu\|\nabla \boldsymbol{g}_1(\boldsymbol{q})^\top \boldsymbol{g}_1(\boldsymbol{q})\|_1$ as described in Section 5.1.

Note that we assumed at the beginning that each $\boldsymbol{g}_i(\cdot)$ represents a an objective function of the trajectory only. Although this was to simplify the reasoning process, this can be further extended easily to the cases where $\boldsymbol{g}_i(\cdot)$ represents a general cost function.

To find a feedback controller stabilizing around the trajectory found by the convergence of the above procedure, we can use the hierarchical SLS described in Section 5.4 around the linearization of that trajectory.

Verification with quadratic costs:

At this stage, it is worth to take a step back to see what this would correspond to in the quadratic objective function case, also for sanity check. We define $\boldsymbol{g}_i(\boldsymbol{u}) = {\boldsymbol{Q}^{\frac{1}{2}}}^{(i)}(\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}) - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)})$, hence $\nabla \boldsymbol{g}_i(\boldsymbol{u}) = {\boldsymbol{Q}^{\frac{1}{2}}}^{(1)} \boldsymbol{S}_{\boldsymbol{u}}$ and

$$\begin{aligned} \Delta \boldsymbol{u}^{(1)} &= -\nabla \boldsymbol{g}_1(\boldsymbol{u}_k)^\dagger \boldsymbol{g}_1(\boldsymbol{u}_k), && (5.27)\\ &= -({\boldsymbol{Q}^{\frac{1}{2}}}^{(1)} \boldsymbol{S}_{\boldsymbol{u}})^\dagger {\boldsymbol{Q}^{\frac{1}{2}}}^{(1)} (\boldsymbol{F}(\boldsymbol{x}_0, \boldsymbol{u}_k) - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}), && (5.28)\\ &= -(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q}^{(1)} \boldsymbol{S}_{\boldsymbol{u}})^\dagger \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q}^{(1)} (\boldsymbol{x}_k - \boldsymbol{\mu}_{\boldsymbol{x}}^{(1)}) && (5.29) \end{aligned}$$

5.6 Experiments

5.6.1 Viapoint reaching

We performed experiments on a 2D double integrator system with the task of reaching a goal position while passing through some viapoints as can be seen in Figure 5.6. The state of the system is composed of the position and the velocity of the point mass as

$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}]$. The primary task is defined as the sum of reaching the green crosses at timesteps 40 and 70 ($\boldsymbol{p}^1_{40}$, $\boldsymbol{p}^1_{70}$), passing through the same viapoints at timesteps 20 and 50, reaching the goal position and zero velocity (G, $\boldsymbol{x}^1_{100}$) at the final timestep: $c_1(\boldsymbol{x}) = \|\boldsymbol{p}_{40} - \boldsymbol{p}^1_{40}\|^2_2 + \|\boldsymbol{p}_{70} - \boldsymbol{p}^1_{70}\|^2_2 + \|\boldsymbol{p}_{20} - \boldsymbol{p}_{50}\|^2_2 + \|\boldsymbol{x}_{100} - \boldsymbol{x}^1_{100}\|^2_2$. The secondary task is the sum of reaching the purple crosses at timesteps 30 and 70: $c_2(\boldsymbol{x}) = \|\boldsymbol{p}_{40} - \boldsymbol{p}^1_{40}\|^2_2 + \|\boldsymbol{p}_{70} - \boldsymbol{p}^1_{70}\|^2_2$. We expect that the controller found by eSLS will do the task at timestep 30, but ignore the task at timestep 70 as this timestep is already occupied by the green viapoint. We solved the following problem using the nullspace method and eSLS as described in Section 5.4

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \boldsymbol{c}_2(\boldsymbol{x}) + \|\boldsymbol{u}\|^2_{\boldsymbol{R}} \\ \text{s.t.} \quad & \boldsymbol{c}_1(\boldsymbol{x}) \quad \text{is minimum.} \\ & \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_0 + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u} \end{aligned} \tag{5.30}$$

and we obtained an eSLS feedback controller in the form of $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{x} + \boldsymbol{k}$. For comparison, we solved a standard eSLS problem with the objective function $c_1(\boldsymbol{x}) + \|\boldsymbol{u}\|^2_{\boldsymbol{R}}$ and obtained the orange trajectory seen in Figure 5.6. Note that it passes through the green viapoints, passes through the same positions at timestep 20 and 50, and reaches the goal position with zero velocity. We then applied the controllers $\boldsymbol{K}, \boldsymbol{k}$ starting from 10 different initial positions and applied these controllers in the presence of noise on the dynamics. First, we see that the noise and the change in the initial position change the position at which the point mass passes through at timesteps 20 and 50, it still passes through the green viapoints and reaches the goal. Second, as expected, the controller robustly passes through the first purple viapoint while ignoring the second one as the timestep 70 is not free.

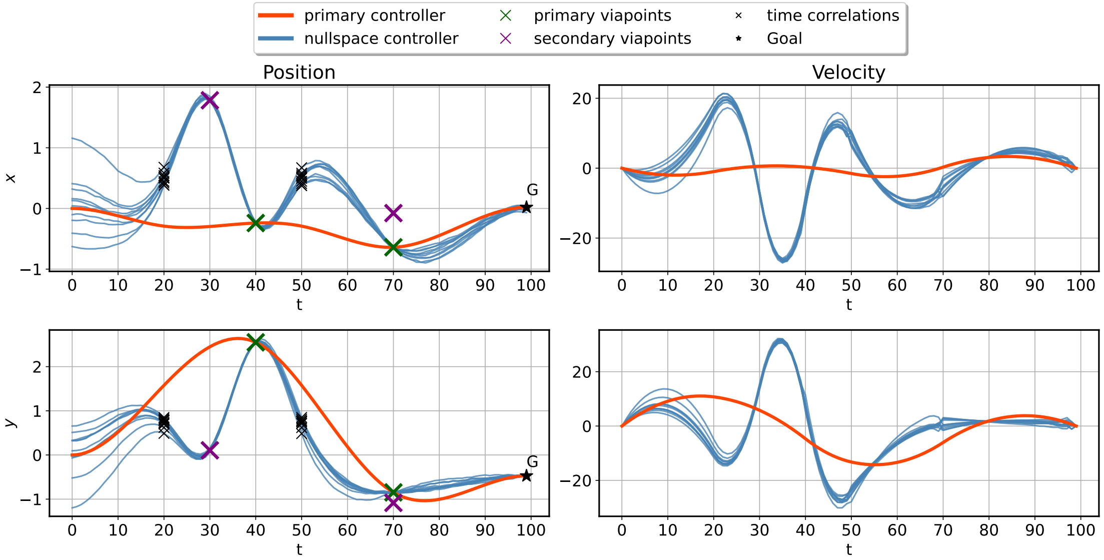


Figure 5.6: 10 realizations (starting from different initial positions) of the primary controller and the nullspace controller

5.7 Conclusion

In this chapter, we tackled the problem of hierarchical optimization for optimal control. We showed that we can set up a bilevel optimization problem and solve it using a linearization of the KKT conditions for the Gauss-Newton method, where each subproblem is solved using nullspace projection operators. We integrated a hierarchical planning strategy with the previously proposed eSLS framework to obtain feedback controllers that are aware of the hierarchies in the task.

Future work consists of analyzing the capabilities of such an optimization problem in terms of initialization, robustness, and computational efficiency. An interesting direction is to combine the proposed method within MPC for the robot to decide automatically which tasks to accomplish respecting their hierarchies.

Learning of Controllers Part II

6 Learning robotic skills from demonstrations

Robotic systems are composed of a feedback loop between the robot and its surroundings. If we could perfectly model everything that connects this loop, including the robot's real-life behavior as a reaction to the changes in the surrounding, and the changes in the surrounding as a reaction to the robot's behavior, we could then have perfect controllers. Feedback control is a way to handle what can not be modeled by reacting to the errors produced by the unmodeled effects. We saw in Chapters 3 to 5 that we can set up optimization problems to find such feedback mechanisms when we have a precise definition of the success of the task as a cost function or constraints, and a good model of the evolution of the state of the task as the robot applies control actions.

In this chapter, we remove the assumptions that we have this expert knowledge of designing cost functions or modeling the task dynamics in a way to be efficiently solved for the particular choice of the solver. Instead, we are interested in two cases: 1. the dynamics model of the task is too complex that the standard solvers fail to find a convergent solution and 2. defining an objective function is much more difficult compared to demonstrating a couple of realizations of the task with different parameters. In both cases, we argue that leveraging learning from demonstration methods significantly improves performance.

In Section 6.1, we exploit human demonstrations in several ways to warm-start and/or guide optimal control problems. We argue that the proposed method indeed saves the solvers from local optima and leads them to the convergence point.

Inputting reference trajectories into a stable controller is a standard way of designing such controllers. The challenge here is to be able to exploit fully the capabilities of the learned reference trajectory model during the execution of the trajectory with the designed controller. In Section 6.2, we propose a method to combine impedance controllers with a probabilistic trajectory model to reconcile the generalization and the multimodality aspects of the model with the robustness of the controller.

6.1 Learning trajectory models to warm-start optimal control

Publication Note

The material presented in this section is adapted from the following publication:

- Xue, T., Girgin, H., Lembono, T. and Calinon, S. (2023). Demonstration-guided Optimal Control for Long-term Non-prehensile Planar Manipulation. In Proc. IEEE Intl Conf. on Robotics and Automation (ICRA).

My contribution concerns the idea of applying a trust region method to DDP to fully exploit the offline feedback and feedforward gains for online execution.

In this section, we focus on long-term non-prehensile planar manipulation, which concerns achieving reliable non-prehensile planar manipulation with a long-term horizon, involving joint logistic and geometric planning and feedback control over diverse interaction modes and face switches. For example, to push an object, a prerequisite is to decide how much force should be applied, and which point to push. Moreover, in some cases such as pushing an object with small distance but large rotation, relying on a single fixed face is not feasible. Therefore, a sequence of face switching is required, as well as contact mode schedule resulting from Coulomb friction.

To achieve non-prehensile planar manipulation, four main challenges appear as follows:
1. **Hybrid Dynamics**. The dynamics of a pusher-slider system depends on the current interaction mode and contact face between the pusher and the slider. In this section, we consider not only the motion involving various interaction modes, e.g., separation, sticking, sliding up, and sliding down, but also the switching between the contact faces, i.e., left, bottom, right, and up (unlike previous works [107, 108] that only work with a single face). The transitions between different faces and the separation to the other three contacting modes pose important difficulty for gradient-based optimization methods.
2. **Underactuation**. The contact force between the pusher and the slider is constrained within a motion cone, which makes it impossible to exert arbitrary acceleration on the object to achieve omnidirectional movement.
3. **Long-horizon TAMP**. Due to the characteristics of hybrid dynamics and underactuation, it is important to reason over long horizon using both logic and geometric descriptors, where the logic variables relate to contact modes and faces and the geometric variables relate to contact points, slider states, switching points, and control commands.
4. **Contact Uncertainty**. Arising from the frictional contact interactions between the pusher and the slider, as well as between the slider and the table, the contact is hard to be modeled precisely, therefore a controller enabling online contact adaptation is required.

To address these challenges, we propose a hybrid framework based on optimization and learning from human demonstrations. An interface is first built to collect human demonstrations of the pushing task. Given a specific target configuration, we use k-nearest neighbor (k-NN) algorithm to retrieve a demonstration trajectory closest to the target configuration, and we use it to formulate the soft constraints of a hierarchical optimal control problem. The optimal control problem is solved offline to produce the optimal trajectory and the optimal control commands. A feedback controller is then used to adapt to the contact uncertainty online by following the offline trajectory.

The main contribution of our work is an approach to solve the complete long-term non-prehensile manipulation task by covering all of the interaction modes and the face switching cases, with the ability of offline planning and online tracking. This is achieved by introducing human demonstration into the optimal control problem. By using demonstrated continuous variables to express the discrete variables implicitly, we can successfully project traditional TAMP formulation into geometric field. A demonstration-guided hierarchical optimization framework is then proposed, allowing the robot to obtain (sub)optimal solutions very quickly. A real-time feedback controller is finally proposed to replan the trajectory, by compensating for model mismatch and contact uncertainty when interacting with the real physical world.

6.1.1 Problem formulation

A optimal control problem (OCP) can be described as

$$\min_{\boldsymbol{u}_t} \quad c_T(\boldsymbol{x}_T) + \sum_{t=1}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{6.1}$$

$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{6.2}$$

where (6.1) is the cost function and (6.2) is the dynamic equation. Practically, due to the nonlinearity and high degrees of freedom involved in robotics, numerical optimization is mostly used to solve this kind of problem. We use DDP [109] in this work, which has been shown effective for this problem [110, 111]. It can also provide local feedback mechanism that was used to keep the robustness of controller in [112].

After minimizing the cost-to-go function w.r.t. $\Delta\boldsymbol{u}_t$, a local stabilizing controller can be obtained as

$$\boldsymbol{u}_t = \hat{\boldsymbol{u}}_t + \boldsymbol{K}_t\left(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t\right) + \boldsymbol{k}_t, \tag{6.3}$$

where $\boldsymbol{K}_t$ is a feedback gain, and $\boldsymbol{k}_t$ is a feedforward term.

Given (6.1) is a non-convex problem, DDP solves it by optimizing around the current solution iteratively. The convergence is very sensitive to the initial guess, which means that it is easy to be stuck at poor local optima if the initial guess is far away from the optimal solution.

6.1.2 Demonstration-started DDP (DS-DDP)

To solve the problem of getting stuck at poor local optima, we introduce human demonstrations into the basic DDP as initialization, which is a popular way for warm-starting OCP [113]. The demonstrations are designed as continuous variables, which can implicitly express the discrete variables instead of explicitly specifying the mode sequence as in previous work [107, 108, 114]. In this way, we convert the joint logic and geometric optimization problem to a typical geometric optimization problem, which can be solved much efficiently.

The collected human demonstrations are denoted as $[\widetilde{\boldsymbol{q}}_s, \widetilde{\boldsymbol{q}}_f, \widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{u}}]$, with $\widetilde{\boldsymbol{q}}_s = [\widetilde{\boldsymbol{q}}_{\boldsymbol{s_0}}, \widetilde{\boldsymbol{q}}_{\boldsymbol{s_1}}, \cdots, \widetilde{\boldsymbol{q}}_{\boldsymbol{s_T}}]$, $\widetilde{\boldsymbol{q}}_f = [\widetilde{\boldsymbol{q}}_{\boldsymbol{f_0}}, \widetilde{\boldsymbol{q}}_{\boldsymbol{f_1}}, \cdots, \widetilde{\boldsymbol{q}}_{\boldsymbol{f_T}}]$, where $\widetilde{\boldsymbol{q}}_{s_t} \in \mathbb{R}^3$ and $\widetilde{\boldsymbol{q}}_{f_t} \in \mathbb{R}^2$ represent the state of the slider and the pusher at timestep t, respectively. $\widetilde{\boldsymbol{v}} = [\widetilde{\boldsymbol{v}}_0, \widetilde{\boldsymbol{v}}_1, \cdots, \widetilde{\boldsymbol{v}}_{T-1}] \in \mathbb{R}^2$ and $\widetilde{\boldsymbol{u}} = [\widetilde{\boldsymbol{u}}_0, \widetilde{\boldsymbol{u}}_1, \cdots, \widetilde{\boldsymbol{u}}_{T-1}] \in \mathbb{R}^2$ denote the velocity and acceleration at each timestep.

Given a target $\boldsymbol{q}_s^*$, we use k-NN to select the index of j^* with the closest demonstration $\widetilde{\boldsymbol{q}}_{s_T}$ to the target in the task space by evaluating

$$j^* = \min_{j \in \mathbb{S}} \quad \text{dist}(\widetilde{\boldsymbol{q}}^j_{s_T}, \boldsymbol{q}_s^*), \tag{6.4}$$

where $\mathbb{S} = \{j : j \in \{0, 1, \cdots, n_d\}$, n_d is the number of demonstrations, and

$$\text{dist}(\boldsymbol{x}, \boldsymbol{y}) = (\sum_{r=1}^{d} |x_r - y_r|^p)^{1/p}, \tag{6.5}$$

where d is the dimension of slider state, and $p = 2$.

The cost function of DS-DDP is defined as:

$$c_1 = c_{re} + c_{rg} + c_{bd}, \tag{6.6}$$

with

$$c_{re} = (\boldsymbol{\mu}_T - \boldsymbol{x}_T)^\top \boldsymbol{Q}_T (\boldsymbol{\mu}_T - \boldsymbol{x}_T), \quad c_{rg} = \sum_{t=0}^{T-1} (\boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t),$$
$$c_{bd} = \sum_{t=0}^{T-1} (\boldsymbol{f}^{\text{cut}}(\boldsymbol{u}_t, \boldsymbol{u}_l)^\top \boldsymbol{K} \boldsymbol{f}^{\text{cut}}(\boldsymbol{u}_t, \boldsymbol{u}_l)),$$

where c_{re} is the reaching cost, and c_{rg}, c_{bd} are the regularizer and boundary penalizer of control commands. $\boldsymbol{u}_l$ is the predefined bounding box of $\boldsymbol{u}$. $\boldsymbol{f}^{\text{cut}}$ is a soft-thresholding function. The initial guess $\boldsymbol{u}^0 = \widetilde{\boldsymbol{u}}$ is drawn from human demonstrations directly.

Although this method seems like an effective warm-starting method, it is restricted by the number of acquired demonstrations.

6.1.3 Demonstration-constrained DDP (DC-DDP)

To alleviate the problem mentioned above, we propose to use demonstration as the soft constraints of control commands in OCP. It is achieved by designing the cost function as

$$c_2 = c_{re} + c_{rg} + c_{bd} + c_{sw} + c_{ve} + c_{ac}, \tag{6.7}$$

with

$$c_{sw} = \sum_{n=t_0}^{t_N} ((\boldsymbol{\mu}_n - \boldsymbol{x}_n)^\top \boldsymbol{Q}_n (\boldsymbol{\mu}_n - \boldsymbol{x}_n)), \tag{6.8}$$

$$c_{ve} = \sum_{t=0}^{T-1} ((\widetilde{\boldsymbol{v}}_t - \boldsymbol{v}_t)^\top \boldsymbol{R}_{dv} (\widetilde{\boldsymbol{v}}_t - \boldsymbol{v}_t)), \tag{6.9}$$

$$c_{ac} = \sum_{t=0}^{T-1} ((\widetilde{\boldsymbol{u}}_t - \boldsymbol{u}_t)^\top \boldsymbol{R}_{du} (\widetilde{\boldsymbol{u}}_t - \boldsymbol{u}_t)), \tag{6.10}$$

where c_{sw}, c_{ve} and c_{ac} are designed to follow the demonstrated face switching strategy, pusher velocity and pusher acceleration. $n = [t_0, \cdots, t_N]$ is the timestep when the contact face switches, $\boldsymbol{\mu} = [\widetilde{\boldsymbol{q}}_s^\top \ \widetilde{\boldsymbol{q}}_f^\top \ \widetilde{\boldsymbol{v}}^\top]^\top$ is the state of the selected demonstration, and $\widetilde{\boldsymbol{v}}_t$, $\widetilde{\boldsymbol{u}}_t$ are the demonstrated velocity and acceleration at timestep t.

6.1.4 Warm-starting DDP (WS-DDP)

Demonstration-constrained DDP shows good performance and can avoid poor local optima, but in order to further improve its convergence properties, we propose a hierarchical optimization framework, where the solution of DC-DDP is used to initialize another DDP problem that we call Warm-starting DDP (WS-DDP).

The cost function of WS-DDP is as same as DS-DDP, allowing it to explore much freely towards the final target. The initial guess $\boldsymbol{u}^0 = \boldsymbol{u}_{DC}^*$ is the result of previous DC-DDP.

6.1.5 Adaptation to disturbance

Similarly to basic DDP, we can see that for a tracking problem, the resulting optimal control policy takes the same form as (6.3), characterized by a feedback gain and a feedforward term. Typically, the optimal control policy is used to generate control commands at each timestep based on the current state to stabilize the motion along the nominal trajectory. However, the frictions and other unmodeled nonlinearities might cause undesired behaviors, especially when the error between the current state and the planned solution $\|\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t\|^2$ is too big. To alleviate this issue, we propose to use an error filtering method based on a trust region defined as a ball of radius r as $\mathcal{B}_t(r) = \{\boldsymbol{x} \in \mathbb{R}^n | \|\boldsymbol{x} - \hat{\boldsymbol{x}}_t\| < r\}$ (as was done in Chapter 3). By denoting the actual robot state as $\boldsymbol{x}_t^0$, we filter the $\boldsymbol{x}_t$ in (6.3) as follows: if $\boldsymbol{x}_t^0 \in \mathcal{B}_t(r)$, then we take $\boldsymbol{x}_t = \hat{\boldsymbol{x}}_t$, else we take $\boldsymbol{x}_t = \boldsymbol{x}_t^0$. This means that if the error between the actual robot state and the planned state is small, we can use the feedforward terms of the controller directly, and if it is big, then we replan the trajectory using the feedback controller gains. This allows us to fully exploit the feedback and feedforward gains computed offline during the online execution of the task avoiding expensive recomputations at each timestep.

6.1.6 Experiments

We evaluate in this section the proposed offline programming method (Sec. 6.1.6) and the proposed online tracking controller (Sec. 6.1.6).

Offline Programming

In this work, the task space is the horizontal plane on the table, and it is limited as $\mathcal{T} = \{[x, y, \theta] : x \in [\text{–}25\text{cm}, 25\text{cm}], y \in [\text{–}25\text{cm}, 25\text{cm}], \theta \in [-\pi, \pi]\}$. We collected 3 representative demonstrations, which are $[15\text{cm}, -10\text{cm}, \text{–}\pi/2], [0, \text{–}20\text{cm}, \pi/2], [15\text{cm}, -15\text{cm}, \pi/2]$, corresponding to $N_s = 0$, $N_s = 1$ and $N_s = 2$, respectively, where N_s is the number of face switches during the demonstration. The initial state is defined as $[0\ 0\ 0\ \alpha p_x\ 0\ 0\ 0]^\top$, where $\alpha = 1.3$, corresponding to the separation mode in Sec. A.1.2, allowing the pusher to select the contact point at the beginning. The cost function gains are set to $\boldsymbol{Q}_T = 10^6 \times \text{diag}\{1, 1, 1, 10^{6p-5}, 10^{1-6p}, 10^{-3}, 10^{-3}\}$, $\boldsymbol{Q}_n = 10^6 \times \text{diag}\{10^{-3}, 10^{-3}, 10^{-3}, p, (1-p), 0, 0\}$, where $p = 1$ when $\theta_f = 0$ or $\theta_f = \pi$, otherwise $p = 0$. $\boldsymbol{R} = \boldsymbol{K} = \text{diag}\{1, 1\}$, $\boldsymbol{R}_{du} = \boldsymbol{R}_{dv} = \text{diag}\{100, 100\}$.

In other works, planar pushing tasks are often formulated as Mixed-Integer Programming (MIP) problems [107, 114, 108]. To compare with our method, we use demonstrations as the initialization for MIP as well, which is called as DS-MIP. CasADi [115] with the Bonmin solver [116] and Crocoddyl [117] with the FDDP solver are used separately to solve DS-MIP and DS-DDP. Fig. 6.1 and Fig. 6.2 show the convergence curve and the solving time of 10 randomly selected target configurations. The vertical axis of Fig. 6.1 is the norm of the reaching error at each iteration with respect to the initial error. We can see in these figures that DS-MIP can almost get the same result as DS-DDP but needs 10 times more time. This is because of the nonconvex nature of integer variables. By implicitly expressing integer variables as demonstrated continuous control commands, TAMP problems can be solved much more efficiently. Moreover, we also find that the proposed DC-DDP and WS-DDP can achieve better results in relatively longer time (but still acceptable) compared to DS-DDP. This shows that simply initializing using the demonstration as done in DS-DDP is not enough, and constraining the search space

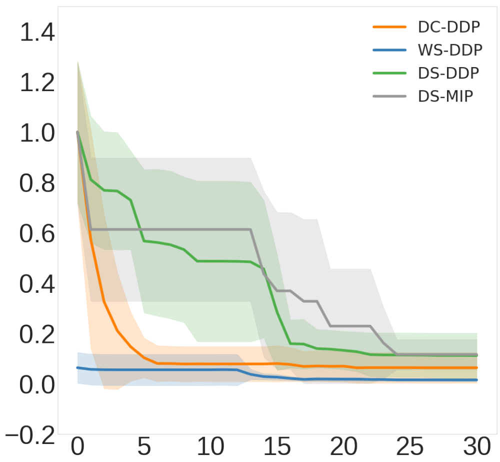


Figure 6.1: Convergence curve

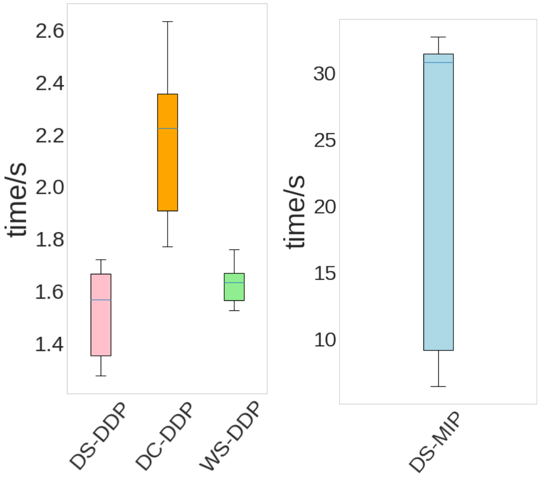


Figure 6.2: Solving time

Table 6.1: Performance of DS-DDP, DC-DDP, and WS-DDP for offline programming

Method	$\boldsymbol{x}_{\text{err}}$/cm	$\boldsymbol{y}_{\text{err}}$/cm	$\boldsymbol{\theta}_{\text{err}}$/rad	succ_rate
DS-DDP	0.24 ± 2.22	0.96 ± 4.03	0.10 ± 0.34	74%
DC-DDP	0.04 ± 1.33	0.85 ± 1.91	0.02 ± 0.08	75%
WS-DDP	**0.11 ± 1.01**	**0.56 ± 1.63**	**0.01 ± 0.07**	**84%**

using demonstrations as the soft constraints during early iterations helps to reach a better solution.

Additionally, we tested the generalization ability of the proposed demonstration-guided offline programming method. A successful offline programming is defined as: $\{x_{\text{err}} < 1\text{cm},\ y_{\text{err}} < 1\text{cm},\ \theta_{\text{err}} < 5°\}$, where x_{err}, y_{err} and θ_{err} are the difference between final pose and target pose. 100 targets are randomly selected in the task space $\mathcal{T}$ to test the generalization capability. The statistical results are listed in Table 6.1. Clearly, with demonstrations, the success rate significantly increases for random-selected targets. By using only 3 demonstrations, DC-DDP can accomplish 75 out of 100 random targets in task space, also with low mean and standard deviations for the errors. WS-DDP achieves a slightly higher success rate based on the DC-DDP solution, showing that this demonstration-guided hierarchical optimization framework can generalize to unknown targets very well. Practically, the generalization result does not change a lot as long as the selected 3 demonstrations are informative. It would be studied in the future about how to collect demonstrations more efficiently and actively.

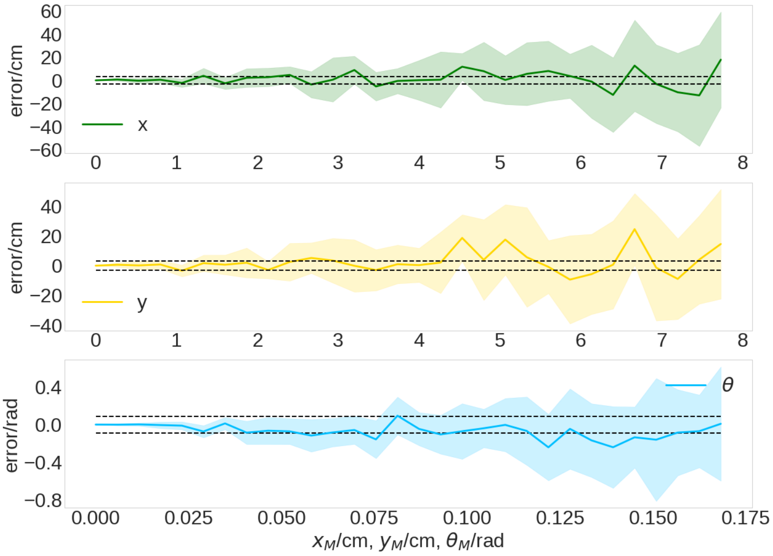


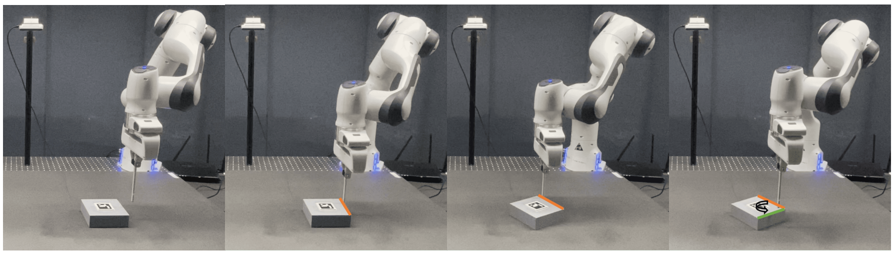
Figure 6.3: Tracking performance under disturbance. The dashed lines present the tolerance.

Online Tracking

For online tracking, we investigated both numerical simulation and real robot experiments. In simulation, we introduce a disturbance on the state as $\boldsymbol{x} = \overline{\boldsymbol{x}} + \boldsymbol{\epsilon}$ from the beginning to the end, where $\epsilon_x \sim \mathcal{U}(-x_M, x_M)$, $\epsilon_y \sim \mathcal{U}(-y_M, y_M)$, $\epsilon_\theta \sim \mathcal{U}(-\theta_M, \theta_M)$, are the components of $\boldsymbol{\epsilon}$, drawn from a uniform distributions. Fig. 6.3 shows the evolution of the errors on x, y, and θ, computed as the difference between the final point and the target, for increasing x_M, y_M and θ_M. The tolerance for online tracking is set as $\{x_{\text{err}} < 3\text{cm},\ y_{\text{err}} < 3\text{cm},\ \theta_{\text{err}} < 5°/0.087\text{rad}\}$. We can find that the controller can successfully resist 4cm perturbation for x and y, and 0.117rad for θ.

Then, we tested the proposed method on the real robot setup (Fig. **??**), using a 7-axis Franka Emika robot and a RealSense D435 camera. The slider ($r_s = 6$cm) is a 3D-printed prismatic object with PLA, lying on a flat plywood surface, with an Aruco Marker on the top face. A wooden pusher ($r_p = 0.5$cm) is attached to the robot to move the object. The motion of the object is tracked by the camera at 30 HZ, and the feedback controller runs at 100 HZ, with a low-level Cartesian impedance controller (1000 HZ) actuating the robot.

In this experiment, we tried one line tracking task with disturbance and two face switching tasks. Both simulated and experimental results achieve the targets within specified tolerance (see accompanying video). The trajectories in Fig. 6.5 correspond to one of the task requiring one face switching to push the object from $[0, 0, 0]$ to $[20\text{cm}, -20\text{cm}, \pi/2]$. The control strategy is intuitive: pushing it from the left face to slightly adjust the pose and then changing to the top face for the next phase of pushing. Fig. 6.5-(a) is the simulation trajectory, which is generated by using PyBullet [118], while


(a) Initialization (b) Contact (c) Pushing (d) Face switching

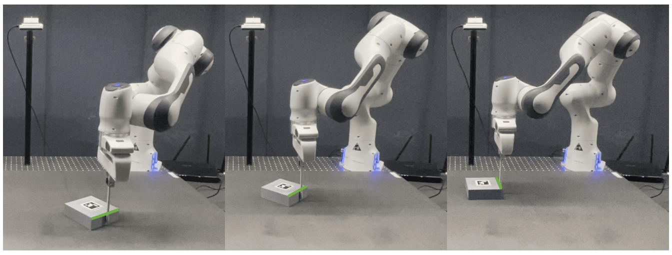
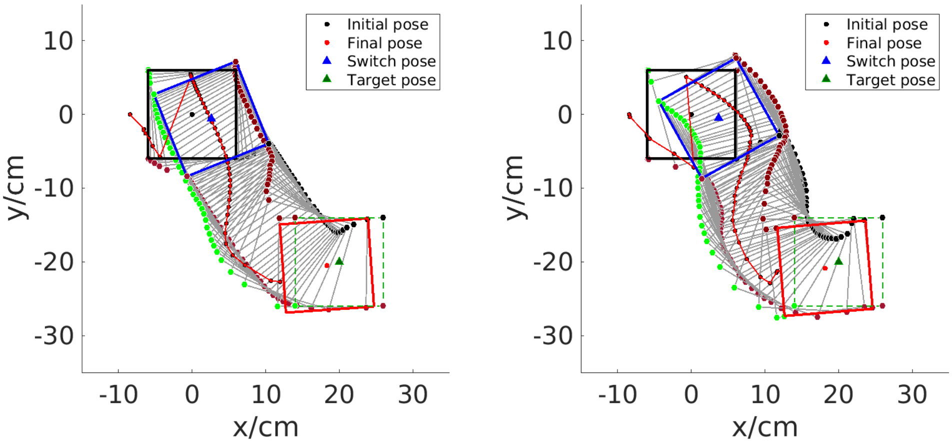

(e) Contact (f) Pushing (g) Reaching

Figure 6.4: Pushing task with face switching. The manipulator starts from (a) and selects an optimal face (orange) and contact point (b) for pushing, until reaching the planned face switching point (c). Next, the manipulator changes to face (d) and touches the object again at the selected point (e), followed by the next phase of pushing (f), until reaching the final target (g). This example is for $N_s = 1$. (d)∼(f) should be repeated if $N_s > 1$. The colored line is used to express the current active face, and the black arrow in (d) represents the face switching process.



a) Simulation results b) Experimental results

Figure 6.5: Planar pushing with face switching. Both simulation and experimental results reach final targets within tolerance.

Fig. 6.5-(b) shows the real robot trajectory. It is observed that these two trajectories are significantly different, because of the different friction parameters in the two different worlds. Still, both can overcome the uncertainty to reach the final target. Despite the existence of unstructured elements such as differences in the visual system and the robot controller, several assumptions of the dynamics model, as well as immeasurable friction, the feedback controller is able to cope with these different mismatches and track the reference trajectory successfully. Fig. 6.4 shows the keyframes of the pusher pushing the slider toward the target, indicating that 7 steps are needed with the face switching strategy. Another final target configuration, $[5\text{cm}, -18\text{cm}, \pi/5]$, which requires two face switches, is additionally shown in the video.

6.1.7 Conclusion

In this section, we propose to add separation modes and face switching mechanisms to the problem of pushing objects on a planar surface. We showed that by introducing human demonstrations, the typical TAMP problem can be expressed as a classical geometric optimization problem, which is much more efficient to be solved. With the proposed demonstration-guided hierarchical optimization framework, we demonstrated significantly better results in terms of generalization and precision compared to the state-of-the-art methods. With more demonstrations, the proposed approach has the potential to precisely reach almost any point in the task space. Additionally, we developed a feedback controller based on DDP feedback gains to replan the trajectory for online tracking. We tested the combination of these approaches in both PyBullet simulation and in real robot application, showing good performance to resist contact uncertainty.

Currently, we are using a feedback controller to track the offline trajectory. If the system is subject to large perturbation such as rotating 180°, an online Model Predictive Control (MPC) may still be required. Nevertheless, our demonstration-guided method is also promising as an optimizer within MPC to avoid poor local optima.

As future work, we aim to apply our demonstration-guided approach to a broader range of manipulation tasks (namely, beyond pushing problems), which requires further study on how to extract constraints from demonstration, and how to formulate an optimal control problem based on the extracted constraints. It would also be relevant to explore extensions of TAMP problems by introducing similar continuous human demonstration. Another future work consists in exploiting human demonstrations in model-based learning strategies to let the robot automatically refine the pushing model and its motion.

6.2 Learning trajectory models for adaptive control

Publication Note

The material presented in this chapter is adapted from the following publication:

- Jankowski, J., Girgin, H. and Calinon, S. (2021). Probabilistic Adaptive Control for Robust Behavior Imitation. IEEE Robotics and Automation Letters (RA-L), 6:2, 1997-2004.

In this publication, my contribution is to partly develop the proposed mathematical model and to help with comparisons with the state-of-the-art.

6.2.1 Probabilistic Movement Primitives

One well-established LfD approach is called probabilistic movement primitives (ProMPs) [119], which permits movement representation and generation. ProMPs have been successfully used for learning different robotic tasks from demonstrations, including rhythmic tasks [120], striking tasks [121], or human-robot collaboration tasks [122]. One of the main capabilities of ProMPs lies in the task generalization, which is usually achieved by conditioning the trajectory distribution to some desired keypoints. It is also desirable and possible to generalize with respect to a context or external variable, which is known before executing the task (such as the mass of an object or the volume of a liquid to pour), by learning the joint distribution of the context variable and the trajectory [123].

A ProMP is a probability distribution over trajectories built from a series of N demonstrations (trajectories) of length T and of D dimensions. A demonstration $\boldsymbol{\tau} \in \mathbb{R}^{(T\times D)}$ is approximated by a sum of M basis functions, which are often chosen as radial basis functions (RBF)

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}\boldsymbol{w}_i + \boldsymbol{\epsilon}, \qquad \text{with} \qquad \boldsymbol{\Phi} = \boldsymbol{\Phi}^{\text{1d}} \otimes \mathbb{I}_D, \tag{6.11}$$

where $\otimes$ represents the Kronecker product, $\boldsymbol{\epsilon}$ is zero-mean i.i.d. Gaussian noise, $\boldsymbol{w}_i$ of size $MD \times 1$ is the weight associated to the i^{th} demonstration, $\boldsymbol{\Phi}^{\text{1d}}_{T\times M}$ is the basis function matrix with $\boldsymbol{\Phi}^{\text{1d}}_{t,m} = \Phi_m(t)$ corresponding to the m^{th} basis function indexed at time t, and $\mathbb{I}_D$ is an identity matrix. The weight vectors associated to each demonstration are computed with least squares as

$$\boldsymbol{w}_i = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\tau}_i. \tag{6.12}$$

A probability distribution $p(\boldsymbol{w})$ can then be learned from the demonstrations $\{\boldsymbol{w}_i\}_{i=1}^N$, usually with a multivariate Gaussian or a GMM.

Control with ProMPs: For a given time t, the corresponding position $\boldsymbol{q}(t)$ and velocity $\dot{\boldsymbol{q}}(t)$ is encoded as

$$\boldsymbol{q}(t) = \boldsymbol{\Phi}(t)\boldsymbol{w}, \quad \dot{\boldsymbol{q}}(t) = \dot{\boldsymbol{\Phi}}(t)\boldsymbol{w}. \tag{6.13}$$

To track the trajectory distribution encoded by the learned probabilistic model $p(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$, Paraschos *et al.* derive a time-varying feedback controller by matching the rolled-out closed-loop system dynamics with the robot state distribution of the ProMP model. Therefore, the system dynamics are linearized, while assuming perfect knowledge of the system dynamics. Thus, unmodeled perturbations may lead to strong deviations from the expected behavior. Furthermore, they do not show how their controller can be extended to the multimodal case or to dynamically changing context variables. In the following, we refer to their controller as *original controller*, which is used as a baseline to benchmark our proposed controller.

More recently, Paraschos *et al.* proposed to learn a joint distribution of the motion trajectory and the recorded control action along the trajectory [124]. The controller is derived by conditioning the control action distribution on the current robot position and velocity to obtain a control policy distribution. They propose to stabilize the controlled system by adding linear feedback terms to let the robot converge back to the mean of the trajectory distribution only if the robot is 'far' from the demonstrated region in the state space. This approach does not require the knowledge of the system dynamics. However, it is limited to the collection of demonstrations only through teleoperation, e.g. by capturing the motion of a human demonstrator and executing the motion on the robot in realtime, to be able to record the control actions.

6.2.2 Formulation of the proposed controller

As an underlying control structure, we propose to use a compliant controller

$$\boldsymbol{u} = -\boldsymbol{K}(\boldsymbol{q} - \boldsymbol{q}_d) - \boldsymbol{D}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + \boldsymbol{M}\ddot{\boldsymbol{q}}_d + \boldsymbol{C}\dot{\boldsymbol{q}}_d + \boldsymbol{g}, \tag{6.14}$$

that enables a robot with the dynamics in Equation (2.8) to track a reference trajectory $\{\boldsymbol{q}_d{=}\boldsymbol{\Phi}(t)\boldsymbol{w},\ \dot{\boldsymbol{q}}_d{=}\dot{\boldsymbol{\Phi}}(t)\boldsymbol{w},\ \ddot{\boldsymbol{q}}_d{=}\ddot{\boldsymbol{\Phi}}(t)\boldsymbol{w}\}_{t=1}^{T}$ with the user-defined positive definite feedback gains $\boldsymbol{K}, \boldsymbol{D} \in \mathbb{R}^{n \times n}$. Note that for a stationary $\boldsymbol{w}$, the closed-loop system controlled by this controller is known to be asymptotically stable. The resulting closed-loop dynamics are linear in the trajectory weights $\boldsymbol{w}$, which can also be interpreted as a latent auxiliary control input. In contrast to the original control formulation, this avoids the necessity of linearizing the passive system dynamics in Equation (2.8) and introduces a stabilizing feedback structure.

As in [125], we model the weight vector $\boldsymbol{w}$ as a random variable that correlates with the demonstrations $\boldsymbol{\mathcal{D}}$ and the current robot state $\boldsymbol{y}$. Thus, we find a probabilistic imitation controller by marginalizing over the weight vectors with

$$p(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) = \int_{\boldsymbol{w}} p(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) d\boldsymbol{w}. \tag{6.15}$$

Paraschos *et al.* derive their controller in a similar fashion [124], however, we propose to exploit the compliant control structure in Equation (6.14) instead of learning a joint distribution of robot states and control actions. The probability distribution $p(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{w})$ focuses all the probability mass at the deterministic controller given in Equation (6.14). Hence, the probability distribution can be written as a Dirac function, such that

$$\begin{aligned} p(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{w}) &= \delta(\boldsymbol{A}\boldsymbol{w} + \boldsymbol{b}), \\ \text{with} \quad \boldsymbol{A} &= \boldsymbol{K}\boldsymbol{\Phi} + (\boldsymbol{C} + \boldsymbol{D})\dot{\boldsymbol{\Phi}} + \boldsymbol{M}\ddot{\boldsymbol{\Phi}}, \\ \boldsymbol{b} &= -\boldsymbol{K}\boldsymbol{q} - \boldsymbol{D}\dot{\boldsymbol{q}} + \boldsymbol{g}. \end{aligned} \tag{6.16}$$

It can be seen that the nonlinear feedback controller is linear w.r.t. the weight vector $\boldsymbol{w}$, which results in tractable control distributions.

The conditional distribution $p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\mathcal{D}})$ of the weight vectors can be reformulated by Bayes' theorem

$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) \propto p(\boldsymbol{y}|\boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{\mathcal{D}}), \tag{6.17}$$

such that the learned ProMP model $p(\boldsymbol{w}|\boldsymbol{\mathcal{D}})$ evolves from the formulation of a probabilistic imitation controller. We complete the probabilistic model in Equation (6.17) by letting the robot state $\boldsymbol{y}$ be correlated with the current weight vector by using a linear Gaussian model $p(\boldsymbol{y}|\boldsymbol{w}){=}\mathcal{N}(\boldsymbol{\Phi}\boldsymbol{w}, \boldsymbol{\Sigma}_{\boldsymbol{q}})\mathcal{N}(\dot{\boldsymbol{\Phi}}\boldsymbol{w}, \boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}})$. Here, the covariance matrices $\boldsymbol{\Sigma}_{\boldsymbol{q}}$ and $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}$ can be understood as a tracking tolerance for the controller w.r.t. a given weight vector $\boldsymbol{w}$.

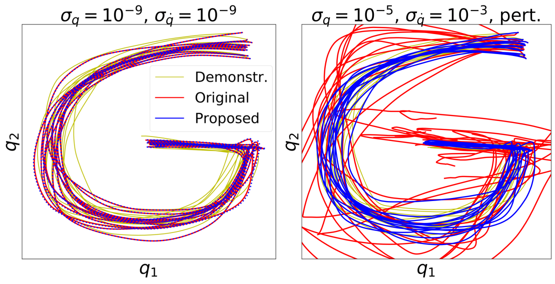


Figure 6.6: Drawing task for a 2D point mass. The parameters of the proposed controller are set to $\boldsymbol{\Sigma}_{\boldsymbol{q}}{=}\sigma_{\boldsymbol{q}}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}{=}\sigma_{\dot{\boldsymbol{q}}}\boldsymbol{I}$, $\boldsymbol{K}{=}500\boldsymbol{I}$ and $\boldsymbol{D}{=}3\sqrt{500}\boldsymbol{I}$. The original controller and the proposed controller produce the same behavior if the tolerance of tracking the probabilistic model is set close to zero (left plot). For the tolerances $\sigma_{\boldsymbol{q}}{=}10^{-5}$ and $\sigma_{\dot{\boldsymbol{q}}}{=}10^{-3}$, the proposed controller adds stabilizing feedback terms. This results in a probabilistic feedback control that is robust to perturbations (right plot).

6.2.3 Single Mode

For simple tasks, e.g. reaching tasks in non-cluttered environments, it is sufficient to learn a single Gaussian distribution of weight vectors $\boldsymbol{w}$ to model the demonstrated movement. In these cases, as also investigated in the original work [125], the learned distribution of weight vectors is given by $p(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$.

As a result, the distribution in Equation (6.17) can be computed closed-form as

$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}), \tag{6.18}$$

where the conditional mean $\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}}$ depends on time and the robot state.

Since the result is a Gaussian distribution, the solution of the integral in Equation (6.15) becomes analytically tractable. Hence, we obtain a state and time-dependent Gaussian distribution of feedback control actions that are conditioned on the demonstrations, namely

$$p(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) = \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}} + \boldsymbol{b},\ \boldsymbol{A}\boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}\boldsymbol{A}^\top), \tag{6.19}$$

with $\boldsymbol{A}$ and $\boldsymbol{b}$ given by Equation (6.16). Obtaining a Gaussian distribution of control actions can be exploited by a *product of experts* scheme in order to blend multiple complementary probabilistic controllers, as shown in [126, 127]. However, in this section, we focus on the imitation performance when using the most likely control action that is given by the mean of the control distribution

$$\boldsymbol{\mu}_{\boldsymbol{u}} = -\tilde{\boldsymbol{K}}(\boldsymbol{q} - \boldsymbol{\Phi}\boldsymbol{\mu}_{\boldsymbol{w}}) - \tilde{\boldsymbol{D}}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{\Phi}}\boldsymbol{\mu}_{\boldsymbol{w}}) + \boldsymbol{M}\ddot{\boldsymbol{\Phi}}\boldsymbol{\mu}_{\boldsymbol{w}} + \boldsymbol{C}\dot{\boldsymbol{\Phi}}\boldsymbol{\mu}_{\boldsymbol{w}} + \boldsymbol{g}. \tag{6.20}$$

The feedback gain matrices are given by

$$\begin{aligned}\tilde{\boldsymbol{K}} &= \boldsymbol{K} - \boldsymbol{A}\boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}\boldsymbol{\Phi}^{\top}\boldsymbol{\Sigma}_{\boldsymbol{q}}^{-1},\\ \tilde{\boldsymbol{D}} &= \boldsymbol{D} - \boldsymbol{A}\boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}\dot{\boldsymbol{\Phi}}^{\top}\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}^{-1}.\end{aligned} \tag{6.21}$$

This shows that the resulting controller has time-varying feedback gains that depend on the variations of the demonstrations.

We validate our analysis and the underlying assumptions in Figure 6.6 by simulating the proposed controller, the original formulation and a controller with constant feedback gains (*Stiff*) in a single-mode drawing task. The left plot shows that the proposed controller with close-to-zero tolerances and the original controller result in a similar behavior. The plot on the right side illustrates the trajectories when there is a random force $\boldsymbol{\tau}_{\text{ext}} \sim \mathcal{N}(\boldsymbol{0}, 10\boldsymbol{I})$ perturbing the system and the tolerances are set to $\sigma_{\boldsymbol{q}}{=}10^{-5}$ and $\sigma_{\dot{\boldsymbol{q}}}{=}10^{-3}$. The proposed controller shows higher robustness to the perturbations than the original controller.

6.2.4 Multiple Modes

For more complicated tasks, it can be desirable to use a more expressive model for the distribution of trajectory weight vectors in order to capture multiple modes. One way to do so is to use a mixture of Gaussians that is a weighted sum of N Gaussian components

$$p(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \sum_{n=1}^{N} \pi_n \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}}^{n}, \boldsymbol{\Sigma}_{\boldsymbol{w}}^{n}), \tag{6.22}$$

where π_n is the mixing coefficient.The conditional distribution of a Gaussian mixture model is itself a Gaussian mixture model given by

$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{\mathcal{D}}) = \sum_{n=1}^{N} \gamma_n(\boldsymbol{y}) \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}}^{n}, \boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}^{n}), \tag{6.23}$$

where $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}}^{n}, \boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}^{n})$ is the conditional distribution of the n-th component and is computed in the same way as Equation (6.18). The new mixing coefficients are given by

$$\gamma_n(\boldsymbol{y}) = \frac{\pi_n p_n(\boldsymbol{y}|\boldsymbol{\mathcal{D}})}{\sum_{l=1}^{N} \pi_l p_l(\boldsymbol{y}|\boldsymbol{\mathcal{D}})}, \tag{6.24}$$

$$\text{with} \quad p_n(\boldsymbol{y}|\boldsymbol{\mathcal{D}}) = \mathcal{N}(\boldsymbol{\Psi}\boldsymbol{\mu}_{\boldsymbol{w}}^{n}, \boldsymbol{\Sigma}_{\boldsymbol{y}} + \boldsymbol{\Psi}\boldsymbol{\Sigma}_{\boldsymbol{w}}^{n}\boldsymbol{\Psi}^{\top}), \tag{6.25}$$

which can be interpreted as the belief of the robot being in mode n. Here, the feature matrix is given by $\boldsymbol{\Psi}{=}[\boldsymbol{\Phi}^{\top}, \dot{\boldsymbol{\Phi}}^{\top}]^{\top}$, and the robot state covariance is given by

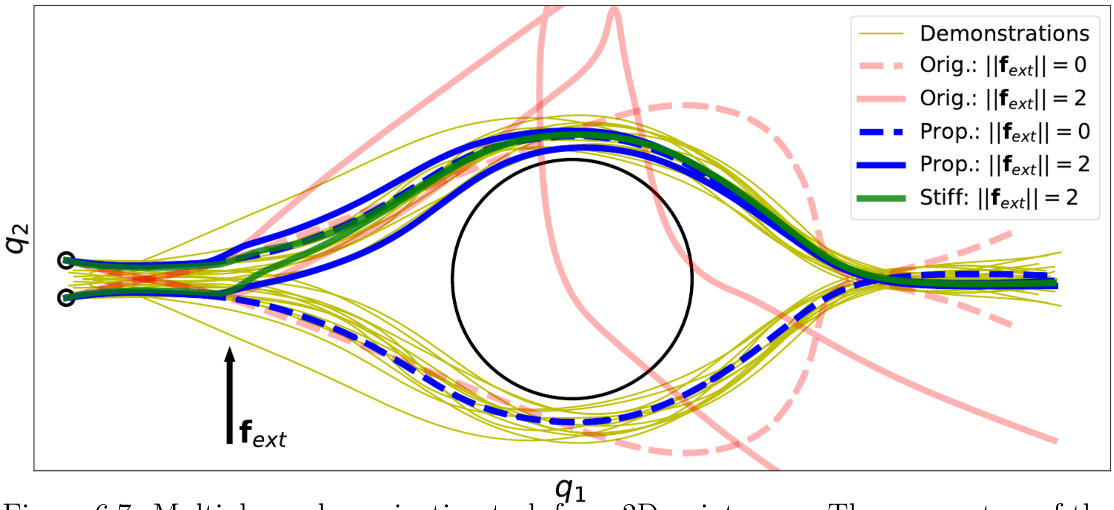


Figure 6.7: Multiple-mode navigation task for a 2D point mass. The parameters of the proposed controller are set to $\boldsymbol{\Sigma_q}$=$10^{-5}\boldsymbol{I}$, $\boldsymbol{\Sigma_{\dot{q}}}$=$10^{-3}\boldsymbol{I}$, $\boldsymbol{K}$=$100\boldsymbol{I}$ and $\boldsymbol{D}$=$3\sqrt{100}\boldsymbol{I}$. The dashed trajectories show the behavior of the original controller (red) and the proposed controller (blue) without perturbations. The solid trajectories show the result of all controllers under the impact of an external force $\boldsymbol{f}_{\text{ext}}$ that pushes the point mass in positive q_2-direction for 0.1 seconds.

$\boldsymbol{\Sigma_y}$=blockdiag($\boldsymbol{\Sigma_q}, \boldsymbol{\Sigma_{\dot{q}}}$). This results in a mixture of Gaussian policies given by

$$\begin{aligned} p(\boldsymbol{u}|\boldsymbol{y},\boldsymbol{\mathcal{D}}) &= \sum_{n=1}^{N} \gamma_n(\boldsymbol{y})\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{u}}^n, \boldsymbol{\Sigma}_{\boldsymbol{u}}^n), \\ \text{with} \quad \boldsymbol{\mu}_{\boldsymbol{u}}^n &= \boldsymbol{A}\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{y}}^n + \boldsymbol{b}, \\ \boldsymbol{\Sigma}_{\boldsymbol{u}}^n &= \boldsymbol{A}\boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{y}}^n \boldsymbol{A}^\top. \end{aligned} \tag{6.26}$$

For the practical control of a robot, it is necessary to find the most likely control action. However, for a Gaussian mixture model, this corresponds to solving a non-convex optimization problem. Due to the time constraints of a realtime control loop, we propose to approximate the control distribution by finding the most likely component based on its mixture coefficient. Consequently, we use the mean control action of the selected component as the control input to the system such that $\boldsymbol{u}$=$\boldsymbol{\mu}_{\boldsymbol{u}}^{n^*}$ with $\gamma_{n^*}(\boldsymbol{y})$>$\gamma_n(\boldsymbol{y})$, $\forall n \neq n^*$.

From a theoretic perspective, the resulting closed-loop system corresponds to a hybrid system, where the function $\gamma_n(\boldsymbol{y})$ represents the guard which indicates the state-dependent switching between multiple closed-loop system dynamics that are all equivalent to the single-mode case that has been discussed in Section 6.2.3.

Figure 6.7 shows the results for a navigation task with two modes that represent possible paths to avoid the collision with the obstacle (black circle). The synthetic demonstrations of the two modes have been separated in advance and the Gaussian components have been computed individually with π_1=π_2=0.5. For the implementation of the original controller and the stiff controller, we use the mixture coefficient computation of our proposed controller, given in Equation (6.24), to find the most likely mode and apply the corresponding control command to the point mass. The dashed signals show the resulting trajectories in the absence of disturbances when started from the initial positions indicated by the small black circles. The solid signals show the resulting trajectories when there is a vertical force $\boldsymbol{f}_{\text{ext}}$ perturbing the point mass during 0.1 seconds. All controllers make use of the two modes in realtime by switching to the upper path after the point mass has been pushed in that direction. The original controller is not able to recover from the short-term perturbation that is caused by the external force and by the mode switching. The stiff controller and the proposed controller show a low tracking error also for the perturbed case.

6.2.5 Feedback of the Context

To encode more general and adaptive skills, it is useful to introduce state-independent context variables that are supposed to affect the behavior of the robot. Context variables are also discussed in the original work of Paraschos *et al.* [125], however only considering the offline adaptation of a single ProMP distribution. Context variables can be used to encode information about the task, e.g. the position and size of the object to pick, but also information about the environment, e.g. the position and size of an obstacle. Since this information may change during the execution of a learned skill, it is desirable to incorporate the context as another random variable in the control policy to generate a reactive behavior. Thus, we reformulate the controller in Equation (6.15) by augmenting the set of conditions by the context $\boldsymbol{s}$ with

$$p(\boldsymbol{u}|\boldsymbol{y},\boldsymbol{s},\boldsymbol{\mathcal{D}}) = \int_{\boldsymbol{w}} p(\boldsymbol{u}|\boldsymbol{w},\boldsymbol{y})p(\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s},\boldsymbol{\mathcal{D}})d\boldsymbol{w}. \tag{6.27}$$

The conditional distribution of the weight vectors $\boldsymbol{w}$ is again given by Bayes' theorem

$$p(\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s},\boldsymbol{\mathcal{D}}) = \frac{p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w},\boldsymbol{s}|\boldsymbol{\mathcal{D}})}{p(\boldsymbol{y},\boldsymbol{s}|\boldsymbol{\mathcal{D}})}. \tag{6.28}$$

Here, the joint probability distribution $p(\boldsymbol{w},\boldsymbol{s}|\boldsymbol{\mathcal{D}})$ can be learned by fitting a single Gaussian distribution as in [125] or by fitting a Gaussian mixture model to the demonstrated data. For the sake of compactness, we directly consider the case of an arbitrary number N of Gaussian components representing the learned joint distribution

$$p(\boldsymbol{w},\boldsymbol{s}|\boldsymbol{\mathcal{D}}) = \sum_{n=1}^{N} \tilde{\pi}_n \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_w\\ \boldsymbol{\mu}_s\end{pmatrix}^n, \begin{pmatrix}\boldsymbol{\Sigma}_w & \boldsymbol{\Sigma}_{ws}\\ \boldsymbol{\Sigma}_{sw} & \boldsymbol{\Sigma}_s\end{pmatrix}^n\right).$$

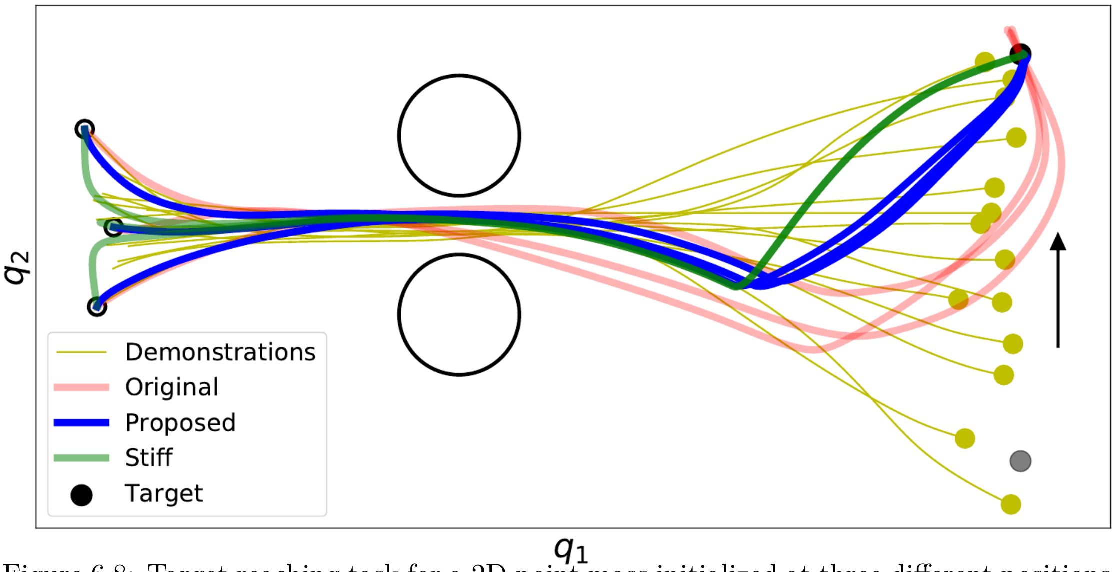


Figure 6.8: Target reaching task for a 2D point mass initialized at three different positions depicted by the small black circles. The parameters of the proposed controller are set to $\boldsymbol{\Sigma}_{\boldsymbol{q}}=10^{-5}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}=10^{-3}\boldsymbol{I}$, $\boldsymbol{K}=100\boldsymbol{I}$ and $\boldsymbol{D}=3\sqrt{100}\boldsymbol{I}$. The target, depicted by the filled black circle, jumps from the position indicated by the grey circle in the lower right corner to the final target in the upper right corner after 2.3 seconds. The duration of the motion is 4 seconds.

Analogously to the multimodality case in 6.2.4, the result of the conditional distribution in Equation (6.28) is given by a mixture of Gaussians

$$p(\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s},\boldsymbol{\mathcal{D}}) = \sum_{n=1}^{N} \gamma_n(\boldsymbol{y},\boldsymbol{s})\mathcal{N}(\boldsymbol{\mu}^n_{\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s}}, \boldsymbol{\Sigma}^n_{\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s}}), \tag{6.29}$$

where the mean of the n-th component additionally depends on the current context variable $\boldsymbol{s}$. The new component coefficient $\gamma_n(\boldsymbol{y},\boldsymbol{s})$ can be viewed as the belief of the robot being in mode n. It is given by

$$\begin{aligned}\gamma_n(\boldsymbol{y},\boldsymbol{s}) &= \frac{\tilde{\pi}_n p_n(\boldsymbol{y}|\boldsymbol{s},\boldsymbol{\mathcal{D}})p_n(\boldsymbol{s}|\boldsymbol{\mathcal{D}})}{\sum_{l=1}^{N}\tilde{\pi}_l p_l(\boldsymbol{y}|\boldsymbol{s},\boldsymbol{\mathcal{D}})p_l(\boldsymbol{s}|\boldsymbol{\mathcal{D}})},\\ \text{with}\quad p_n(\boldsymbol{y}|\boldsymbol{s},\boldsymbol{\mathcal{D}}) &= \mathcal{N}(\boldsymbol{\Psi}\boldsymbol{\mu}^n_{\boldsymbol{w}|\boldsymbol{s}}, \boldsymbol{\Sigma}_{\boldsymbol{y}} + \boldsymbol{\Psi}\boldsymbol{\Sigma}^n_{\boldsymbol{w}|\boldsymbol{s}}\boldsymbol{\Psi}^\top),\end{aligned}$$

where the marginalized context distribution is given by $p_n(\boldsymbol{s}|\boldsymbol{\mathcal{D}})=\mathcal{N}(\boldsymbol{\mu}^n_{\boldsymbol{s}}, \boldsymbol{\Sigma}^n_{\boldsymbol{s}})$.

Similarly to Section 6.2.4, we propose to approximate the Gaussian mixture distribution by the most likely component and to use the corresponding mean control action, such that $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{\mu}^{n^*}_{\boldsymbol{w}|\boldsymbol{y},\boldsymbol{s}} + \boldsymbol{b}$ with $\gamma_{n^*}(\boldsymbol{y},\boldsymbol{s}) > \gamma_n(\boldsymbol{y},\boldsymbol{s})$, $\forall n \neq n^*$.

Figure 6.8 illustrates the behavior of a 2D point mass for a single-mode goal reaching task where the target is jumping during the execution of the task. For the implementation of the original controller, we used its control formulation and replaced the stationary ProMP distribution parameters $\boldsymbol{\mu}_{\boldsymbol{w}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ by the context-conditional ProMP distribution parameters $\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{s}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{w}|\boldsymbol{s}}$. The *stiff* controller tracks the mean $\boldsymbol{\mu}_{\boldsymbol{w}|\boldsymbol{s}}$ of the conditional distribution with the constant feedback gains $\boldsymbol{K}$ and $\boldsymbol{D}$. In addition to the synthetic trajectory demonstrations, a target position, depicted by the yellow circle, has been recorded as a context variable $\boldsymbol{s}{=}\boldsymbol{q}_{target}$. During the simulation, the context changes by a jump from the gray filled circle to the black filled circle at $t{=}2.3$s. The full simulation duration is $t{=}4$s.

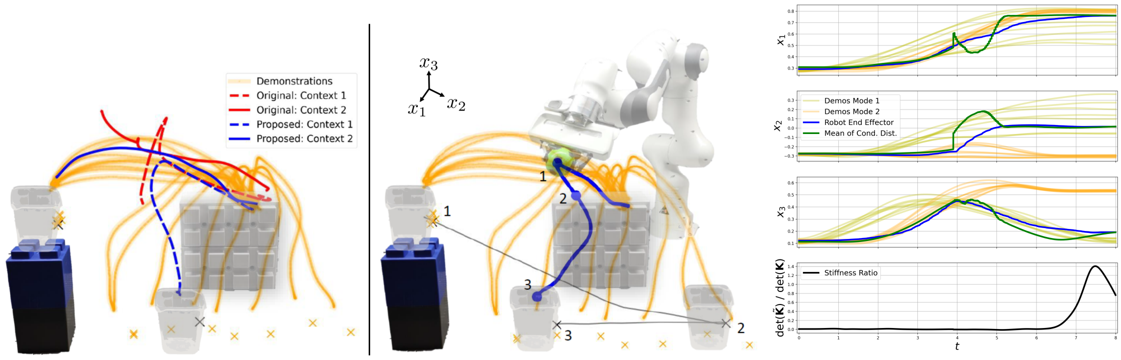


Figure 6.9: The Franka Emika robot has to drop a ball into the box using the presented imitation controller (blue). The skill is demonstrated (orange trajectories) for a given box position (orange crosses). Experiment 1 (*left*): The original controller (red) and the proposed controller are tested on two different static box positions (context 1 and 2). Experiment 2 (*center* and *right*): The proposed controller is tested in a dynamic scenario, where the context variable changes during the execution (i.e. the box is moving from position 1 to 3) such that the position reference (green) changes accordingly. The proposed controller adapts its stiffness based on the variability of the demonstrations and based on the correlation of the trajectory phase with the context variable (indicated by the stiffness ratio).

6.2.6 Experimental Validation

We conduct the experiments using a 7-axis Franka Emika Panda robot, by using the mathematical model of the system dynamics in Equation (2.8). As this model does not include parasitic, nonlinear effects such as joint friction, these appear as inherent perturbations during the execution. This is the case in many robotic platforms and thus model-based controllers should be able to cope with these perturbations. The objective of the experiments is to show that our proposed controller can achieve this robustness while imitating reactively the demonstrated behavior and exploiting variations in realtime.

We consider the task of placing a ball inside a box in a cluttered environment. The task is fulfilled if the ball has been dropped into the box that is moving during the execution without colliding with the environment. Figure 4.1 shows the experimental setup, including the initial robot pose in the lower-left corner. The box position is detected by a stereo vision system and is used as a context variable. We provide 16 demonstrations of the robot end-effector trajectory using kinesthetic teaching. In Figure 6.9, each demonstrated trajectory is a solution to the task for a given context value (box position), depicted by yellow crosses. Note that the context values are fixed during each demonstration, such that tracking a moving box has not been demonstrated explicitly. We manually separate the demonstrations into two modes. The first mode encodes solutions for situations where the box is on the table with some variations, while the second mode encodes solutions when the box is placed on the blocks on the left with no variations.

We compute two individual ProMP models according to Section 6.2.5 by using the separated demonstrations. We then combine the individual models to obtain a Gaussian mixture model by computing the mixture coefficients according to the number of demonstrations provided for each mode. Each of the two components uses 12 radial basis functions as trajectory features. We implement the original formulation using an inverse dynamics approach as described in [125].

The parameters of the proposed controller are selected as $\boldsymbol{\Sigma}_{\boldsymbol{q}}{=}10^{-5}\boldsymbol{I}$, $\boldsymbol{\Sigma}_{\dot{\boldsymbol{q}}}{=}10^{-3}\boldsymbol{I}$, $\boldsymbol{K}{=}10^3\boldsymbol{I}$ and $\boldsymbol{D}{=}3\sqrt{10^3}\boldsymbol{I}$. To resolve the kinematic redundancy of the robot for both controllers, we implement a compliant regulator in the nullspace of the end-effector task using the initial configuration as a reference. Both approaches run at a control frequency of 1 kHz.

Figure 6.9 illustrates the results of the experiments (see also the accompanying video). In the left plot, we show a comparison between the proposed controller and the original controller for a box position that does not change during the execution. The dashed lines depict the behavior of the original controller and the proposed controller for the lower box position, while the corresponding solid lines show the behavior for the upper left box position. We can see that both controllers manage to generate motions without the robot colliding with the environment. However, the original controller does not move the robot to the required position, and the robot fails to put the ball into the box because of the inherent perturbations that are not part of the model that both controllers are based on. Our proposed controller, on the other hand, manages to move the robot to successfully drop the ball into the box, for both box positions, as also indicated by the evaluation of the tracking error in Table 6.2.

Table 6.2: Metric-based comparison for the ball-in-box task (in Figure 6.9).

Method and Context	**Tracking Error**
Original, Context 1	319.45
Proposed, Context 1	0.46
Original, Context 2	520.76
Proposed, Context 2	0.71

In the second experiment, we evaluate the capability of the proposed controller to adapt robustly to dynamic changes of the context variable, i.e. the box position. The center plot and the right plot in Figure 6.9 shows the resulting end-effector trajectory produced by the proposed controller. During the execution of the task, the box is moved from position 1 (as labeled in Figure 6.9) to position 2 then 3, as depicted by the black curve in the center plot. The corresponding markers on the robot trajectory roughly indicate the end-effector positions at that time. The mean of the conditioned trajectory distribution represents the reference of the controller. It can be seen that the movement of the box results in jerky changes of the reference around t=4s, which would be tracked stiffly by a controller with constant feedback gains. The plot in the lower-right corner of Figure 6.9 illustrates the ratio between the determinant of the varying stiffness gain $\tilde{\boldsymbol{K}}$ and the tuned constant stiffness gain $\boldsymbol{K}$. It shows that the controller learned to use a higher stiffness as the correlation between the context and the trajectory phase increases. In this case, the correlation grows towards the end of the trajectory as the context mainly affects the final position of the end-effector. In summary, the proposed controller solves the given task by switching from the second mode to the first mode after moving the box from the upper left position to a lower position. The resulting trajectory shows that the controller produces smooth transitions between the two modes, together with a smooth adaptation to the changing box position by exploiting the demonstrated variations.

6.2.7 Conclusion

In this section, we derived a stochastic feedback controller by imposing a compliant control structure on the latent trajectory feature variable $\boldsymbol{w}$ and conditioning the control action on the demonstrated behavior. We showed that the original controller proposed in [125] is similar to a limit case of our proposed controller. We analyzed the robustness of the closed-loop system depending on the parameters of the proposed controller and compared the results with the original controller. Furthermore, we showed that our proposed controller readily extends to multiple modes and to the adaptation to dynamic context variables in realtime. We evaluated the theoretical hypotheses in simulated and real-world experiments with the Franka Emika robot. We showcased in these experiments that our proposed controller outperforms the original controller in terms of robustness, and that this property is a key to the successful reproduction of demonstrated movements with robots characterized by unmodeled dynamic effects (typically, joint friction).

In future work, we plan to investigate practical limitations of the presented approach, e.g. finding computational bottlenecks and testing tasks with more than two concurring modes. We also plan to extend the proposed approach to the problem of robustness to context variables that are far from the demonstrated distribution and against intentional physical interactions to consider human-robot interaction in a principled way (e.g. to safely switch to a different mode by interacting with the robot). Furthermore, we plan to investigate adaptation mechanisms for the phase variable of the reference distribution in order to relax the time-dependency of the proposed controller.

7 Active learning of feedback controllers

Learning from demonstration (LfD) offers an intuitive framework to overcome the difficulty of programming robots by teaching them movements using an adaptive representation. In LfD, the demonstrations are often acquired by kinesthetic teaching or by teleoperation. One of the main advantages of these techniques is that they allow non-expert users to easily (re-)program the robots by generalizing the models to different tasks. This requires a set of demonstrations to provide various executions of the task, whose acquisition is often costly. Thus, we want to collect these demonstrations in an efficient manner. Often, non-expert users struggle to identify what demonstration will be the most informative to the robot [128]. One way to alleviate this limitation is to provide the user with some feedback, such as a visual illustration of what the robot has currently learned [129]. Yet, such an approach requires the appropriate design of a feedback mechanism, which might not be trivial in a high-dimensional task, and still requires the user to choose the demonstration eventually.

Active learning is a promising approach to address the aforementioned issues. An active learning framework develops and tests new hypotheses in an interactive learning process. In robotics, the robot is first provided with initial demonstrations from which an initial model of the task can be built. Then, at each stage of the active learning framework, the robot requests a new demonstration in order to improve the model. This contrasts with passive learning systems that attempt to explain the model only according to available training data. Ideally, the robot should request the new demonstration around a query point that will maximize the information gain. Specifically, the information gain is related to the part of the input space where the model uncertainties are the highest [17].

In robotics applications, two different kinds of uncertainties arise, namely *(i) aleatoric uncertainties* and *(ii) epistemic uncertainties.* The aleatoric uncertainties represent the variations in the demonstrations and are typically used to adapt the behavior of the robot, e.g. its compliance at different phases of the task. In contrast, the epistemic uncertainties are related to the lack of knowledge (i.e. data) in the demonstrations and is typically used for informative exploration. In other words, aleatoric uncertainties cannot be reduced by adding more data, while epistemic uncertainties can be. For this reason, the quantification of epistemic uncertainties is crucial for active learning frameworks. A natural way to take these uncertainties into account is through Bayesian inference [130].

In this chapter, we propose an active learning approach to improve the generalization capabilities of control and trajectory policies in a behavior cloning setup. Our approach is based on the Bayesian inference method in [37] which models a joint state-action distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with Bayesian Gaussian mixture models (BGMMs). The conditional (predictive) distribution of the policy $p(\boldsymbol{u}_t|\boldsymbol{x}_t)$ is then found by conditioning on the current state $\boldsymbol{x}_t$. In [37], the authors use a product of experts (PoE) framework to exploit the uncertainties inherent to Bayesian models to fuse several control policies (see Section 7.2 for a brief background).

In Section 7.3, we extend the work in [37] to apply active learning based on an information density function. We first propose a decomposition of the covariance matrix of the posterior BGMM distribution into aleatoric and epistemic parts in Section 7.3.1. The quadratic Rényi entropy is then used to compute the related uncertainties of Gaussian mixture models (GMMs) in closed form (see Section 7.3.2). As explained in Section 7.3.3, the next query point of our active learning framework is obtained by maximizing an information-density cost based on the quadratic Rényi entropies. In particular, we propose to approximate this cost with a GMM to represent highly uncertain region distribution. This notably avoids local optima problems during uncertainty maximization. Finally, we demonstrate the efficiency of our approach on a reaching task in a cluttered environment in a 2D simulated example and with a real experiment on a Panda robot (see Section 7.3.4). The experiment setup is presented in Figure 7.5a.

One limitation of the method presented in Section 7.3 is that the uncertainties are computed for an action given the current state. Hence, it is not applicable to robotic tasks where one needs to reason about the uncertainty over the whole task (e.g., over the whole trajectory), which is often the case in robotics (for instance for object grasping, assembly or navigation tasks). Also, the method requires the possibility to start and show a demonstration from any given state, which is not always possible (for instance, starting a demonstration in the middle of a dynamic throwing task or a pouring task is not feasible).

To address the aforementioned limitations, in Section 7.4, we present an active learning framework based on a joint trajectory-context distribution $p(\boldsymbol{\tau}, \boldsymbol{s})$ learned from demonstrations. We propose an active learning approach for ProMPs and quantify separately aleatoric and epistemic uncertainties in ProMP the same way as in Section 7.3 using BGMM. We demonstrate the applicability of our approach in Section 7.4.1 on three different pouring task experiments. The first two experiments are performed in simulation to allow quantitative comparisons and for reproducibility purposes. The last experiment shows the applicability of the approach on a real 7-DoF robot pouring task.

7.1 Related Work

A collection of recent work focuses on improving and fine-tuning learned movement representations using reinforcement learning (RL) [48, 49] and iterative learning control (ILC) [50]. As these methods iteratively minimize a reward function, LfD can be used to determine the initial point of the optimization in order to favor a safe exploration. In contrast, information-theoretic explorations in behavior cloning methods have been exploited only in few works to enhance the quality and the generalization abilities of the learned movement models [18, 51, 19].

One of the simplest and widely used active learning methods is uncertainty sampling. Using an uncertainty measure, the robot is expected to request a query point in the most uncertain region of the input space. If the model can only encode aleatoric uncertainties, one can train several probabilistic representations with different local convergence properties. The disagreement between each individual model and their average model is then maximized using KL divergence as explained in [17]. Other techniques consist in reducing the variance of error in a regression problem. In general, this is intractable. Simplifications occur by using Fisher Information and Cramér-Rao inequality as in [52]. All the aforementioned methods are myopic as they only care about the information content of single data instances. This can result in models selecting outliers or exploring far away in the context space where no generalization is required. Information-density methods overcome this problem by choosing instances that have high information content and are still representative of the underlying distribution. This is achieved by using a weighted product of uncertainty measure (entropy, ensemble, etc) and similarity measure (Euclidean distance, correlation coefficients, etc.) [17].

As the data acquisition process is usually costly in robotics, active learning has emerged as a viable solution. It has been shown that active learning permits a faster exploration of the action space [131], which is particularly true in the context of developmental robotics, where active learning is often referred to as curiosity-driven learning [132, 133]. In the context of learning from demonstrations, active imitation learning [134] is a topic gaining interest. It has indeed been successfully used in a variety of robotic tasks, such as autonomous navigation [135, 51]. In [136], the authors leverage the uncertainties on a discrete hypothesis space to request meaningful demonstrations to a human teacher. Several approaches have also been proposed in the context where the learner does not request full demonstrations, but only the action to take at a given state [134, 137]

In [18], the authors use Gaussian Process Regression (GPR) in a reaching task to map object positions to the weights encoding the trajectories via Dynamic Movement Primitives (DMP). They demonstrate an active learning framework based on the GPR epistemic uncertainties for reaching to a predefined set of object positions to improve their DMP model. They work with time-dependent trajectory policies without control information. They measure the epistemic uncertainty of a whole trajectory given a context, while aleatoric uncertainties (variations) are not considered. Our work differs from [18] in two ways. First, as we consider state-dependent policies including both aleatoric and epistemic uncertainties. Second, their approach in [18] exploits uncertainty sampling, which would diverge if the uncertainty is defined over a continuous variable instead of a discrete set of variables. To overcome this problem, we use information-density methods.

In [19] the authors propose an active learning method for learning ProMPs. The distribution is learned in the ProMP weight space using a GMM. They then use the marginal distribution over the internal context space (trajectory keypoint) to request demonstrations for contexts that are the furthest from any Gaussian (as Mahalanobis distance). Their approach is evaluated for a reaching task where different grasps are possible, with attempts to generalize over different poses of the object. This approach has several limitations. First, they choose the next context to query based only on some distance in the context space. While in their application this can make sense since the contexts (keypoints) are closely correlated with the trajectory distributions, this is not relevant for a more general external context. Indeed, representing the context space well is not so useful, as our ProMPs are used to generate trajectory distributions for a given context. Rather, what matters is whether a given context influences the trajectory distribution. In this regard, their method would aim to represent a context variable with no influence on trajectories equally well as other more meaningful context variables. In contrast, our method focuses on the conditional distribution of the weights given the context, hence learning dependencies and correlations between the context variables and the movement. A second limitation is that the use of a GMM does not take into account epistemic uncertainties but only aleatoric ones, while work in active learning [17] has shown that metrics based on aleatoric uncertainties are less effective than those based on epistemic uncertainties. Lastly, their approach uses a heuristics to add Gaussians during learning using a threshold. Indeed, the Mahalanobis distance does not depend on the weights attributed to the different Gaussians, which might bias the learning towards unlikely portions of the context space. In contrast, we use Bayesian inference to infer the number of Gaussians using a Dirichlet prior on the mixing coefficients.

7.2 Background

In this section, we present the BGMM framework exploited to learn control policies presented in [37]. As state-dependent control policies learned with BGMM can create unstable behaviors, the BGMM policy is fused with another stable control policy within the PoE framework.

7.2.1 Bayesian Gaussian Mixture Model

In this section, the Bayesian analysis of a Gaussian Mixture Model (GMM) is treated following [130]. Let $\boldsymbol{x} = \left[\boldsymbol{x}^{i\top} \ \boldsymbol{x}^{o\top}\right]^\top \in \mathbb{R}^D$ be the joint observation of the input and the output with dimension $D = D_i + D_o$. The joint distribution is defined with a mixture of K multivariate normal distributions (MVNs) with means $\boldsymbol{\mu}=\{\boldsymbol{\mu}_k\}$, precision matrices $\boldsymbol{\Lambda}=\{\boldsymbol{\Lambda}_k\}$ and mixing coefficients $\boldsymbol{\pi}=\{\pi_k\}$ as

$$p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}).$$

We define a latent variable $\boldsymbol{z}$, each component of which is a binary variable $z_k \in \{0, 1\}$ such that $\sum_{k=1}^{K} z_k = 1$. We can associate the mixing coefficients to the latent variables with $p(z_k{=}1) = \pi_k$ so that $p(\boldsymbol{z}|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k}$. We then obtain $p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\Lambda}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_k}$. The conditional distributions $p(\boldsymbol{Z}|\boldsymbol{\pi})$, $p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$, the conjugate prior distributions $p(\boldsymbol{\mu}, \boldsymbol{\Lambda})$ and $p(\boldsymbol{\pi})$ of the joint observation dataset $\boldsymbol{X}=\{\boldsymbol{x}_n\}$ and the latent variable dataset $\boldsymbol{Z}=\{\boldsymbol{z}_n\}$ are summarized in Table 7.1.

Table 7.1: conditionals and priors where $\mathcal{W}(\cdot)$ and $\mathrm{Dir}(\cdot)$ correspond to Wishart and Dirichlet distributions

Conditional of $\boldsymbol{X}$ $p(\boldsymbol{X}\|\boldsymbol{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda})$	$\prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_n\|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})^{z_{nk}}$
Conditional of $\boldsymbol{Z}$ $p(\boldsymbol{Z}\|\boldsymbol{\pi})$	$\prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}}$
Prior on $\boldsymbol{\mu}, \boldsymbol{\Lambda}$ $p(\boldsymbol{\mu}, \boldsymbol{\Lambda})$	$\prod_{k=1}^{K} \mathcal{N}\Big(\boldsymbol{\mu}_k\|\boldsymbol{m}_0, (\beta_0 \boldsymbol{\Lambda}_k)^{-1}\Big) \mathcal{W}(\boldsymbol{\Lambda}_k\|\boldsymbol{W}_0, \nu_0)$
Prior on $\boldsymbol{\pi}$ $p(\boldsymbol{\pi})$	$\mathrm{Dir}(\boldsymbol{\pi}\|\alpha_0)$

As explained in [130], closed-form update equations for Expectation-Maximization (EM) algorithm is derived by using a factorized variational distribution. Note that EM update equations are usually implemented in machine learning libraries such as *scikit-learn* for Python.

For robotic applications, we determine the predictive density of a new observation point $\hat{\boldsymbol{x}} = \left[\hat{\boldsymbol{x}}^{i\top} \ \hat{\boldsymbol{x}}^{o\top}\right]^\top$ equivalent to a mixture of multivariate t-distributions with mean $\hat{\boldsymbol{m}}_k$, covariance matrix $\hat{\boldsymbol{L}}_k$, mixing coefficients $\hat{\pi}_k$ and degree of freedoms $\hat{\nu}_k$ as [130]

$$p(\hat{\boldsymbol{x}}|\boldsymbol{X}) = \sum_{k=1}^{K} \pi_k \mathrm{t}(\hat{\boldsymbol{x}}|\boldsymbol{m}_k, \boldsymbol{L}_k, \nu_k), \tag{7.1}$$

where

$$\begin{aligned}
\pi_k &= \frac{\alpha_k}{\sum_{k=1}^{K} \alpha_k}, \tag{7.2}\\
\nu_k &= \nu_k + 1 - D, \tag{7.3}\\
\boldsymbol{L}_k &= \frac{(\nu_k + 1 - D)\beta_k}{1 + \beta_k} \boldsymbol{W}_k, \tag{7.4}\\
\boldsymbol{m}_k &= \bar{\boldsymbol{m}}_k. \tag{7.5}
\end{aligned}$$

with the update equations on α_k,β_k ν_k, $\boldsymbol{W}_k$ and $\bar{\boldsymbol{m}}_k$ are given in [130]. We can then define the distribution of the output conditioned on the input as

$$p(\hat{\boldsymbol{x}}^o|\hat{\boldsymbol{x}}^i, \boldsymbol{X}) = \sum_{k=1}^{K} \pi_k^o|it(\hat{\boldsymbol{x}}^i|\boldsymbol{m}_k^o|i, \boldsymbol{L}_k^o|i, \nu_k^o|i), \tag{7.6}$$

where

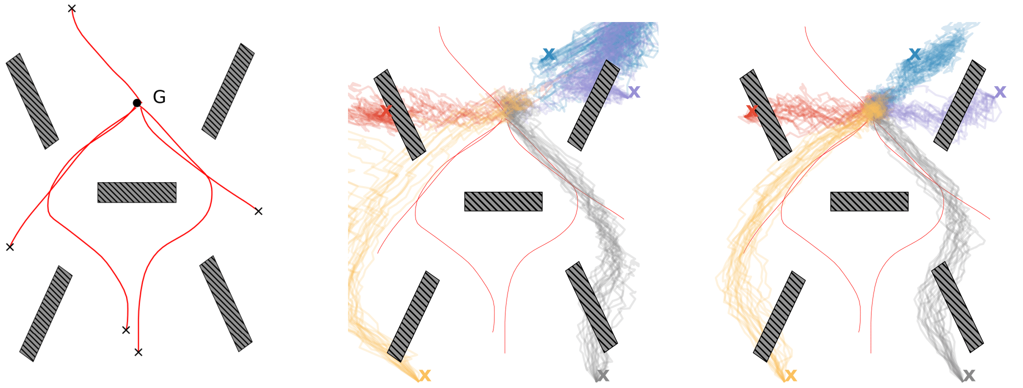
$$\begin{aligned}
\pi_k^o|i &= \frac{\pi_k \mathrm{t}(\hat{\boldsymbol{x}}^i|\boldsymbol{m}_k^i, \boldsymbol{L}_k^i, \nu_k^i)}{\sum_{j=1}^{K} \pi_j \mathrm{t}(\hat{\boldsymbol{x}}^i|\boldsymbol{m}_j^i, \boldsymbol{L}_j^i, \nu_j^i)}, \tag{7.7}\\
\nu_k^o|i &= \nu_k + D^i, \tag{7.8}\\
\hat{m}_k^o|i &= \boldsymbol{m}_k^o + \boldsymbol{L}_k^{oi} \boldsymbol{L}_k^{ii^{-1}} (\hat{\boldsymbol{x}}^i - \boldsymbol{m}_k^i), \tag{7.9}\\
\boldsymbol{L}_s &= \boldsymbol{L}_k^{oo} - \boldsymbol{L}_k^{oi} \boldsymbol{L}_k^{ii^{-1}} \boldsymbol{L}_k^{oi^{\top}}, \tag{7.10}\\
L_k^o|i &= \frac{\nu_k + (\hat{\boldsymbol{x}}^i - \boldsymbol{m}_k^i)^{\top} \boldsymbol{L}_k^{ii^{-1}} (\hat{\boldsymbol{x}}^i - \boldsymbol{m}_k^i)}{\nu_k^o|i} \boldsymbol{L}_s. \tag{7.11}
\end{aligned}$$

In this work, we consider the input $\hat{\boldsymbol{x}}^i$ and the output $\hat{\boldsymbol{x}}^o$ equivalent to the state $\boldsymbol{x}$ and the control command $\boldsymbol{u}$, respectively. Note that the stability of this controller is determined by the positive-definiteness of the term $\boldsymbol{L}_k^{oi} \boldsymbol{L}_k^{ii^{-1}}$. To guarantee the controller stability, the PoE framework is introduced in the next section.

7.2.2 Product of Experts

Robot movements learned with state-action abstractions result in probabilistic controllers with no guarantee of stability, unless explicitly constrained to be stable as in [35]. To overcome this problem, we fuse the probabilistic unstable controller with another probabilistic stable controller which acts as an attractor towards the demonstration area when the uncertainty in the unstable controller is high. We refer to this fusion of controllers as a *product of experts* (PoE), where each expert represents a stochastic controller with different uncertainty properties. Note that many types of controllers with different uncertainties can be fused to work in parallel. For more details, we refer the reader to [37].

In this work, the stabilizing controller is defined as a probabilistic linear quadratic tracker policy, which can be expressed as a MVN. It can be viewed as a controller which attracts the system to the demonstrated regions when the BGMM controller is very uncertain. When the BGMM control policy is a GMM, the fusion or PoE is defined as the product of a GMM and a MVN, which results in another GMM policy. As an illustrative example, consider a 2D reaching task in a cluttered environment. Fig. 7.1a displays the initial demonstrations starting from different initial positions (cross) to reach goal position ($\boldsymbol{G}$). We choose 5 different random test initial positions and reproduce the trajectories by sampling from a BGMM model and a PoE model. The resulting trajectories are shown in Fig. 7.1b and 7.1c, respectively. Even though the trajectories are more stable in 7.1c (notice that some of the trajectories in 7.1b diverge), the task cannot be accomplished without colliding with the obstacles. In this case, supplementary demonstrations are necessary, and active learning permits to collect them in an informed way.



(a) Initial demonstrations (b) Policy samples from BGMM (c) Policy samples from PoE

Figure 7.1: *(a)* Demonstrations and *(b)-(c)* reproductions of a reaching task in a cluttered environment. The goal position is denoted by G and the obstacles are represented as dashed rectangles. The demonstrated trajectories are depicted with red lines. The policy samples acquired from the BGMM and PoE are depicted by colored lines.

7.3 Active Learning of Control Policies

Publication Note

The material presented in this chapter is adapted from the following publication:

- Girgin, H., Pignat, E., Jaquier, N. and Calinon, S. (2020). Active Improvement of Control Policies with Bayesian Gaussian Mixture Model. In Proc. of IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS), pp. 5395-5401.

Control policies are defined as the probability distribution of control commands or actions

$\boldsymbol{u}$ given the state $\boldsymbol{x}$, denoted as $p(\boldsymbol{u}|\boldsymbol{x})$. They encode the demonstration trajectories along with the dynamics information of the controlled system. As described in Section 7.2.1, we impose a BGMM model structure for the control policy and estimate the parameters of the predictive conditional distribution from the demonstrations.

In this section, we present the proposed active learning of control policies approach. First, a cost function is defined using the epistemic uncertainties in the BGMM control policy and optimized while considering a soft constraint to be a on the desired region of the state-space. The robot then asks for a new demonstration around the query point found by the optimization process. The data of the new demonstration is added to the previous dataset and the BGMM parameters are updated. The robot iterates this process until it reaches a predefined percentage of uncertainty reduction.

In order to build the active learning cost function, the covariance matrices of the control policy must be decomposed into its aleatoric and epistemic parts (Section 7.3.1). Then, we deploy Rényi entropy to calculate epistemic uncertainties in closed-form (Section 7.3.2). The complete formulation of the resulting cost is presented in Section 7.3.3.

7.3.1 Uncertainty decomposition

The uncertainty in the posterior distribution of the BGMM model encodes the variations in the demonstrations, called aleatoric uncertainty, along with the epistemic uncertainty, measuring the lack of knowledge of the model. These different uncertainty modalities are depicted in Fig. 7.2 for our illustrative example. In active learning, we are interested in increasing the knowledge of the model, by providing demonstrations around interesting regions of input space.

In BGMM model, the covariance matrix of the conditional posterior predictive distribution of (7.11) can be decomposed into aleatoric and epistemic parts as

$$\hat{\boldsymbol{L}}_k^o|i = \hat{\boldsymbol{L}}_k^{\text{aleatoric}} + \hat{\boldsymbol{L}}_k^{\text{epistemic}} \tag{7.12}$$

where

$$\hat{\boldsymbol{L}}_k^{\text{aleatoric}} = \frac{\hat{\nu}_k}{\hat{\nu}_k^o|i}\boldsymbol{L}_k^{oo} - \boldsymbol{L}_k^{oi}{\boldsymbol{L}_k^{ii}}^{-1}{\boldsymbol{L}_k^{oi}}^{\top}, \tag{7.13}$$

$$\hat{\boldsymbol{L}}_k^{\text{epistemic}} = \frac{(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{m}}_k^i)^{\top}{\boldsymbol{L}_k^{ii}}^{-1}(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{m}}_k^i)}{\hat{\nu}_k^o|i}\boldsymbol{L}_k^{oo} - \boldsymbol{L}_k^{oi}{\boldsymbol{L}_k^{ii}}^{-1}{\boldsymbol{L}_k^{oi}}^{\top}, \tag{7.14}$$

Notice that the aleatoric uncertainty does not depend on the input point $\hat{\boldsymbol{x}}^i$, while the epistemic uncertainty is a quadratic function of $\hat{\boldsymbol{x}}^i$. The former represents the variability and the noise in the demonstrations and the latter encodes the uncertainty caused by finite data. In robotics, both types of uncertainty are important to capture, i.e. the variations of the demonstrations and the uncertainty in the model, for applications such as compliance adaptation and active learning.

7.3.2 Rényi entropy of the posterior distribution

When the posterior distribution $p(\boldsymbol{u}|\boldsymbol{x})$ is a multivariate GMM (or can be approximated by one), the information-theoretical Shannon entropy does not admit an analytical form. In order to avoid a significant amount of computational burden for the minimization of active learning cost, we use instead the quadratic Rényi entropy, which admit a differentiable closed form for GMMs [138]. Another reason is that it is very close to Shannon entropy value as will be detailed below.

A random variable $\boldsymbol{U}$ from a multivariate t-distribution $\boldsymbol{U} \sim t_\nu(\boldsymbol{u}|\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x}))$ can be approximated by a multivariate normal distribution with mean $\tilde{\boldsymbol{\mu}}(\boldsymbol{x})$ and covariance $\tilde{\boldsymbol{\Sigma}}(\boldsymbol{x})$ using moment-matching method, so that

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{x}) = \boldsymbol{\mu}(\boldsymbol{x}), \qquad \tilde{\boldsymbol{\Sigma}}(\boldsymbol{x}) = \frac{\nu}{\nu-2}\boldsymbol{\Sigma}(\boldsymbol{x}).$$

This approximation can be extended to mixtures using the same mixing coefficients. The Rényi entropy of order α is defined as $H_\alpha(p) = \frac{1}{1-\alpha}\log\int p^\alpha(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$ with $\alpha > 0$ and $\alpha \neq 1$. In the limit case where $\alpha \to 1$, the Rényi entropy is equivalent to the Shannon entropy defined as $H_\alpha(p) = -\int p(\boldsymbol{x})\log p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. In this section, we propose to use quadratic Rényi entropy defined as

$$H_2(p(\boldsymbol{u}|\boldsymbol{x})) = -\log\int p^2(\boldsymbol{u}|\boldsymbol{x})\mathrm{d}\boldsymbol{u}$$

since it admits a closed-form expression for GMMs. Note that the Rényi entropy is a non-increasing function of α, so that $H_1(.) > H_2(.)$. In an active learning framework, the entropy can be used as an uncertainty measure to minimize by searching for the queries that have high entropy values. Even though the Shannon entropy is usually used in information theory, maximizing the quadratic Rényi entropy is equivalent to maximizing a lower bound of the Shannon entropy, which would also maximize it suboptimally. The quadratic Rényi entropy for a posterior distribution represented as a GMM $p(\boldsymbol{u}|\boldsymbol{x}) = \sum_{k=1}^K \pi_k(\boldsymbol{x})\mathcal{N}(\boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x}))$ can be expressed as [138]

$$H_2(p(\boldsymbol{u}|\boldsymbol{x})) = -\log\sum_{i=1}^K\sum_{j=1}^K \pi_i(\boldsymbol{x})\pi_j(\boldsymbol{x})e^{\Delta_{ij}(\boldsymbol{x})}, \tag{7.15}$$

where

$$\Delta_{ij} = \frac{1}{2}\Big(\boldsymbol{\mu}_{ij}\boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{\mu}_{ij} - (\boldsymbol{\mu}_i^\top\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i + \boldsymbol{\mu}_j^\top\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\mu}_j) - \log\frac{|\boldsymbol{\Sigma}_i^{-1} + \boldsymbol{\Sigma}_j^{-1}|}{|\boldsymbol{\Sigma}_i^{-1}||\boldsymbol{\Sigma}_j^{-1}|} - d\,\log 2\pi\Big) \tag{7.16}$$

for the i^{th} and j^{th} components of a GMM, with $\boldsymbol{\Sigma}_{ij} = (\boldsymbol{\Sigma}_i^{-1} + \boldsymbol{\Sigma}_j^{-1})^{-1}$ and $\boldsymbol{\mu}_{ij} =$

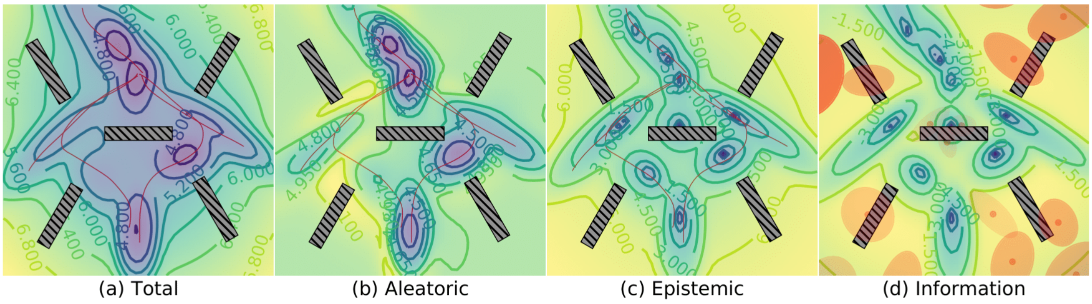

(a) Total (b) Aleatoric (c) Epistemic (d) Information

Figure 7.2: Uncertainty colormaps of the learned control policy for a reaching task in a cluttered environment. *(a)*, *(b)* and *(c)* show the total, aleatoric and epistemic uncertainties of the BGMM, respectively. High to low uncertainties are depicted by colors ranging from yellow to purple. *(d)* depicts the information-density cost and the Gaussian components of the GMM model approximating this cost.

$\boldsymbol{\Sigma}_{ij}(\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i + \boldsymbol{\Sigma}_j^{-1}\boldsymbol{\mu}_j)$.

Fig. 7.2 depicts *(a)* the total, *(b)* aleatoric and *(c)* epistemic uncertainties computed via the quadratic Rényi entropy of the BGMM model of our illustrative example (Fig. 7.1a). Yellow and purple colors depict high and low uncertainties, respectively. Note that the uncertainty of the aleatoric model stays constant as we move away from known data, while it increases in epistemic model. As the epistemic model describes unseen regions, it must be used for an efficient search in the state-space.

7.3.3 Information-density cost for active learning

Following a similar approach to information weighted technique described in Section 7.1, we constrain the optimization space by adding a similarity function that measures the closeness to a region of space where we want to improve our model. In this work, we represent this region as a probabilistic density function (pdf). Note that, even though the region of interest may often be represented as a uniform distribution, one may want to favor some parts of this region compared to others using other distributions. Therefore, we can solve the following optimization problem

$$\arg\min_{\boldsymbol{x}} -H_2(p(\boldsymbol{u}|\boldsymbol{x})) - \beta \log p_{\text{sim}}(\boldsymbol{x}), \tag{7.17}$$

with the epistemic cost in closed-form, to find the next query point $\boldsymbol{x}$, where β is a variable weighing the relative importance of the costs. In practice, uniform distributions will result in negative infinity log probabilities in the outside regions and will not have a defined gradient at the border. Therefore, we approximate the uniform distribution by an MVN using the same mean and diagonal covariance matrix to alleviate this issue. Another problem with the optimization of Eq. (7.17) is the existence of flat regions from which the optimization cannot escape. To overcome this problem, we propose to approximate the epistemic cost in Eq. (7.17) as a GMM with a variational distribution $q(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})$ to represent all the regions where epistemic uncertainty is high, using reverse KL divergence as in

$$\operatorname*{arg\,min}_{\boldsymbol{x}} KL\Big(q(\boldsymbol{x})||H_2(p(\boldsymbol{u}|\boldsymbol{x})) + \beta \log p_{\text{sim}}(\boldsymbol{x})\Big). \tag{7.18}$$

Note that one can also augment the epistemic cost defined in Eq. (7.17) with other costs (see robotic experiment in Section V.B.), so that $q(\boldsymbol{x})$ can represent a more constrained space (e.g. being away from an undesirable region). We can obtain the next query point either by sampling from $q(\boldsymbol{x})$ or by taking the mean of one of the components. As we add more demonstrations and improve our model using this query point, the optimization in Eq. (7.18) can be started with the parameters of the previous $q(\boldsymbol{x})$, which would increase convergence speed. We expect a decrease of entropy in $q(\boldsymbol{x})$ at each iteration of active learning. This gives us a natural way of monitoring the uncertainty reduction.

Fig. 7.2*d* shows the information density colormap favoring to be inside of the figure frame where we want to generalize our model. It also shows the GMM contour ellipses (with 1 standard deviation) which approximate the high information-density regions (yellow). The transparency reflects the mixing coefficient of the GMM. We can observe that the highly uncertain regions are well approximated.

7.3.4 Experiments

Illustrative reaching task

We use the proposed active learning framework to gather iteratively 10 more demonstrations for our illustrative 2D reaching task. At each step, the model informs the teacher on the next query point, given by the mean of the GMM component with the highest mixing coefficient (corresponding to the highest uncertainty). As any sample from that component can be used as a next query point, the closest feasible position to the mean can be chosen if the mean does not correspond to a feasible location, e.g. if it collides with the obstacles.

Note that we are interested in reducing the epistemic uncertainties in the conditional model, which is a function of the input point as in (7.14). In order to define an entropy reduction, we need a measure that does not depend on the input point. We can thus measure how much the entropy changes via the GMM model which approximates highly uncertain regions. Fig. 7.3*a-(top)* shows the evolution of the quadratic Rényi entropy of the GMM model across the active learning iterations. Red crosses show the current entropy values, whereas the black curve is 2D polynomial fit to these values. We can observe that the entropy of the GMM is reduced until there is no component left which can specialize on certain regions with small covariance (small covariance means low entropy). After 6 iterations, the entropy starts to increase as the components are more diffused with bigger covariance matrices. We generally observed that the entropy of the GMM behaves similarly to the the black curve in Fig. 7.3*a-(top)*. The evolution of the entropy of the marginal model $p(\boldsymbol{x})$ is represented in Fig. 7.3*b-(top)*. As expected, the entropy of the marginal model decreases with the quantity of data. Therefore, it results in no explicit method to infer the convergence of the learning process. In contrast, with our GMM model, one can argue that the system has learned a significant percentage of the unseen regions after 6 iterations.

We conducted 5 more experiments performing active learning where new random demonstrations are provided for 5 iterations. The mean and standard deviation of 5 experiments at each iteration are shown in Fig. 7.3*a-(bottom)* for the GMM model and in Fig. 7.3*b-(bottom)* for the marginal model. This demonstrates that the random exploration is not guaranteed to reduce the epistemic uncertainties, even in the marginal model.

The resulting reproductions from the chosen random initial test positions using samples from the updated BGMM and PoE policies are shown in Fig. 7.4a and Fig. 7.4b, respectively. We observe that both policies successfully avoids all the obstacles in the average, while using PoE framework results in a more stable system. The query points of each iteration of active learning are also labeled in Fig. 7.4a. We observe that these query points are rather intuitive, as they correspond to locations that could be chosen by a human to better teach the task to the robot. However, informative query points may be very difficult to choose in other cases where the query space is not easily interpretable.

Robot Experiment

We investigate the reaching in a cluttered environment task shown in Figure 7.5a within our active learning framework. The main challenge of this task is to place the cup inside the white box without colliding with the environment and without pouring the cup. The robot can place the cup from any open side of the box, as long as the cup is inside. Planning methods can be applied to find a joint configuration trajectory starting from a given initial configuration of the robot without colliding with the environment, given the size and positions of the obstacles. However, learning control policies using BGMM offers the advantage of sampling the next state much faster than standard planning methods. It also provides a formal way of improving the planned trajectory using active learning framework proposed in this work. For the improvement of the learned policy, it is difficult for the teacher to choose informative joint configurations intuitively as the demonstrations can take place starting from many different end-effector positions, which correspond to many more joint configurations. Our goal in this experiment is to show that our method provides “intuitive” and informative query points in the joint space of the robot.

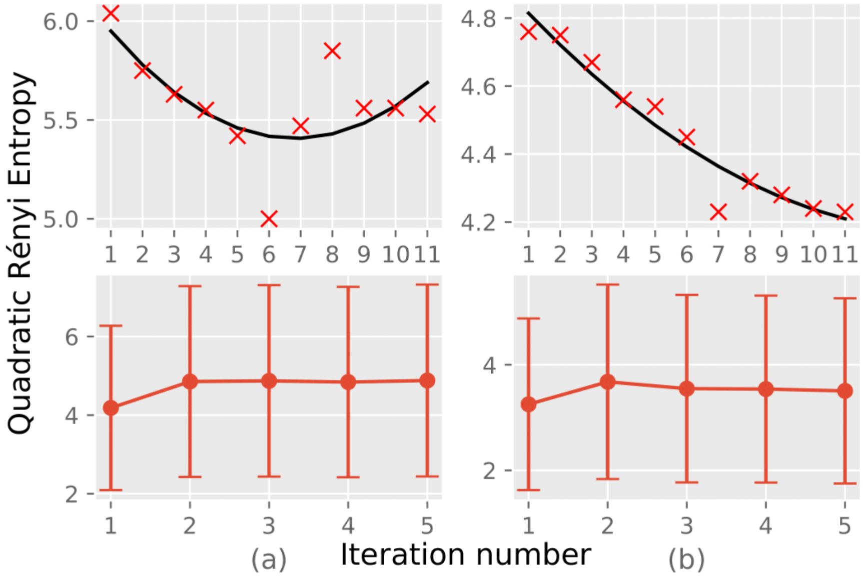
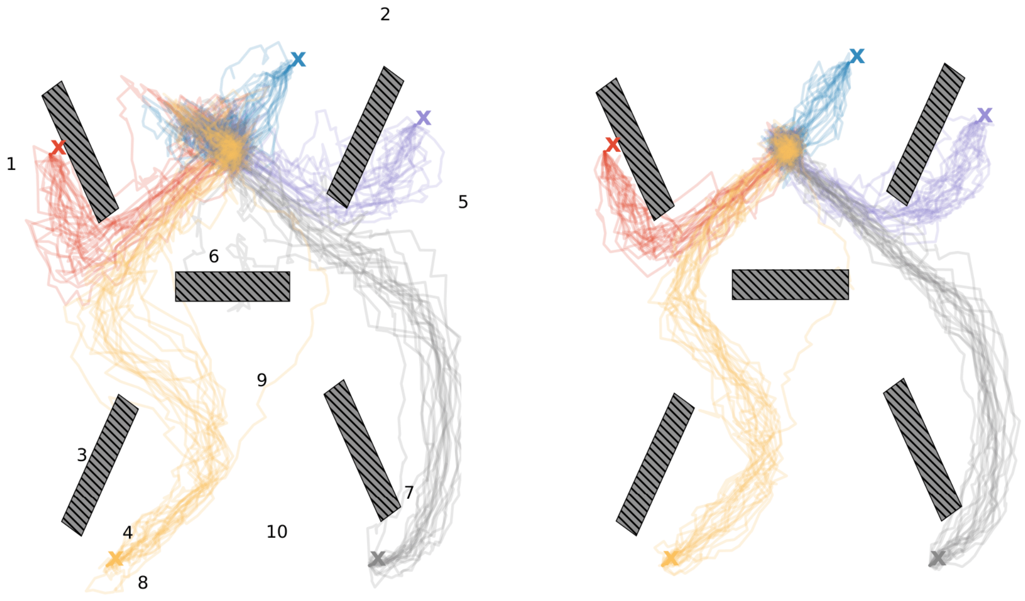


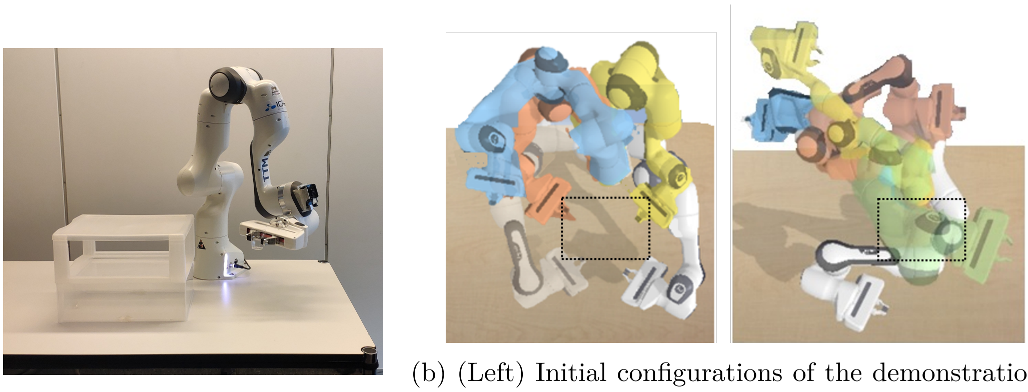
Figure 7.3: Evolution of the quadratic Rényi entropy of (a) the GMM model that approximates highly uncertain regions and (b) the marginal BGMM model. Top figures represent the evolution for the proposed active learning, while the error bars in bottom figures show the mean and the standard deviation of 5 different random exploration for 5 iterations.



(a) Policy samples from BGMM

(b) Policy samples from PoE

Figure 7.4: Reproductions of the learned policy after 10 iterations of active learning. The numbers on (a) denotes the location the query point at each iteration of active learning.



(a) Experimental setup with Franka Emika Panda robot.

(b) (Left) Initial configurations of the demonstrations, (Right) Requested initial configurations for demonstration

Figure 7.5: The task is to put the cup inside a box covered from top and bottom, starting from anywhere in the space. The robot has to maintain a specific end-effector orientation to perform the task without pouring the cup. The main challenge is not to collide with the box and the other obstacles in the environment.

We first demonstrate the reaching task from 11 different initial configurations and learn our control policy. Note that the demonstrations are taken from each side of the box, where it was easy to perform kinesthetic teaching. The initial configurations of the demonstrations are depicted in Figure 7.5b (left).

To improve the model, one need to start from a rather different and informative initial configuration of the arm, which is not easy. Note that the robot has to maintain upright position of the cup to place it inside the box without pouring it and without colliding with the environment. That is why the search space we are interested in is constrained such that we add 2 more cost functions to Eq. (7.17) in the form of probability distributions: *i*) a cost to keep orientation with respect to x-y axis of the robot base fixed and *ii*) a cost to be within the joint limit range of the robot as

$$p(\boldsymbol{x}) = H_2(p(\boldsymbol{u}|\boldsymbol{x})) + \beta \mathrm{f}_{\text{limits}}(\boldsymbol{x}) + \alpha \log p_{\text{upright}}(\boldsymbol{x})$$

where $\mathrm{f}_{\text{limits}}(\cdot)$ is typically the sum of lognormal cumulative distribution functions representing inequality functions which represent the joint limits and $p_{\text{upright}}(\cdot)$ is a normal distribution on the quaternion describing upright orientation in the manifold.

We approximate this cost by a GMM of 10 components minimizing KL divergence between $q(\boldsymbol{x})$ and $q(\boldsymbol{x})$ as in Eq. 7.18. The resulting query configurations (samples from GMM) are given in Figure 7.5b (right). We can see that our GMM could in fact approximate highly uncertain and unseen configurations of the robot as it requests demonstrations around these regions. These configurations are also within the joints range of the robot, and maintain approximately a fixed x-y axis orientation so that the robot will keep the cup upright, without pouring. Although showing that the usefulness of encoding aleatoric uncertainties here is out of scope of this section, it has been exploited in the previous work in [37]. Since the aperture size of the sides are big enough, one can imagine exploiting high variations in the demonstrations while the end-effector enters one side of the box. The learned model would create compliant control commands in these areas which would help the teacher to correct the robot movement during a failing execution. Note that GPR could not encode aleatoric uncertainties.

7.4 Active Learning of Trajectory Policies

Publication Note

The material presented in this chapter is adapted from the following publication:

- Kulak, T., Girgin, H., Odobez, J.-M. and Calinon, S. (2021). Active Learning of Bayesian Probabilistic Movement Primitives. IEEE Robotics and Automation Letters (RA-L), 6:2, 2163-2170.

My contribution in this section is the development of mathematical formulae for the application of the BGMM models in ProMP.

The goal of the task lies in how to modulate the movement represented as ProMP weights $\boldsymbol{w}$ based on different contexts $\boldsymbol{s}$. We denote the context space $\mathcal{C}$ as the space of all possible contexts we would like our robot to be able to generalize to. Formally, this means that there exists an unknown ground truth target distribution $p^{\mathrm{GT}}(\boldsymbol{s}, \boldsymbol{w})$ that can be used to generate robot movements $p^{\mathrm{GT}}(\boldsymbol{w}|\boldsymbol{s})$ adapted for context $\boldsymbol{s}$.

A common way [32, 123] to take into account context variables is to learn the joint distribution of contexts and weights $p(\boldsymbol{s}, \boldsymbol{w})$, where $\boldsymbol{s}$ is the context variable of size D^s. For notation convenience, we introduce $\tilde{\boldsymbol{w}}_i = [\boldsymbol{s}_i^\top, \boldsymbol{w}_i^\top]^\top$, hence $p(\boldsymbol{s}, \boldsymbol{w}) = p(\tilde{\boldsymbol{w}})$. The most general and common uncertainty measure is the Shannon entropy [139]. Initially proposed for discrete random variables, the Shannon entropy has been extended to continuous probability distributions, in which case it is called continuous (or differential) entropy. We propose to quantify the uncertainty of our conditional ProMP by calculating the (continuous) entropy of its epistemic part.

The entropy of a mixture of multivariate t-distributions cannot be obtained analytically. To avoid computationally expensive Monte Carlo sampling methods, we propose to approximate the distribution with a GMM, for which there is a closed-form lower bound of the entropy. The epistemic part of the conditional ProMP distribution can be approximated by a mixture of K Gaussians using moment matching:

$$\tilde{\pi}_k(\boldsymbol{c}) = \hat{\pi}_k^o|i,\ \tilde{\boldsymbol{\mu}}_k(\boldsymbol{c}) = \hat{\boldsymbol{m}}_k^o|i,\ \tilde{\boldsymbol{\Sigma}}_k(\boldsymbol{c}) = \frac{\hat{\nu}_k^o|i}{\hat{\nu}_k^o|i - 2}\hat{\boldsymbol{L}}_k^{\mathrm{ep}}(\boldsymbol{c}). \tag{7.19}$$

We propose to use the closed-form lower bound introduced in [140], which has been shown to be tight. It is expressed as (for clarity purposes we omit the fact that all GMM parameters depend on $\boldsymbol{c}$)

$$H_{lower}\Big(p^{ep}(\hat{\boldsymbol{x}}^o|\hat{\boldsymbol{x}}^i, \boldsymbol{X})\Big) = \frac{1}{2}\Big(K \log 2\pi + K + \sum_{i=1}^{K} \tilde{\pi}_i \log |\tilde{\boldsymbol{\Sigma}}_i|\Big) - \sum_{i=1}^{K} \tilde{\pi}_i \log \sum_{j=1}^{K} \tilde{\pi}_j e^{-C_\alpha(p_i,p_j)}, \quad (7.20)$$

where $C_\alpha(p_i, p_j)$ is the Chernoff α-divergence distance function between the i^{th} and j^{th} Gaussians for $\alpha \in [0, 1]$:

$$C_\alpha(p_i, p_j) = \frac{(1-\alpha)\alpha}{2}(\tilde{\boldsymbol{\mu}}_i - \tilde{\boldsymbol{\mu}}_j)^\top \Big((1-\alpha)\tilde{\boldsymbol{\Sigma}}_i + \alpha\tilde{\boldsymbol{\Sigma}}_j\Big)^{-1}(\tilde{\boldsymbol{\mu}}_i - \tilde{\boldsymbol{\mu}}_j) \quad + \frac{1}{2}\log\left(\frac{|(1-\alpha)\tilde{\boldsymbol{\Sigma}}_i + \alpha\tilde{\boldsymbol{\Sigma}}_j|}{|\tilde{\boldsymbol{\Sigma}}_i|^{1-\alpha}|\tilde{\boldsymbol{\Sigma}}_j|^{\alpha}}\right). \quad (7.21)$$

In practice we choose $\alpha = 1/2$, in which case the Chernoff divergence is the Bhattacharyya distance. Finding the context which maximizes the epistemic entropy can be done either using a grid search if the context space is of low dimension, or using a Bayesian optimization algorithm.

7.4.1 Experiments

In this section, we evaluate our active learning method in four different ways related to the pouring task. The first three favor quantitative results and reproducibility by using a simulated environment and a given database of demonstrations to choose from. In the last experiment, we consider the pouring task on a real 7 DoF Franka Emika robot.

In all experiments, we use $N = 20$ evenly spread Gaussian radial basis functions (RBFs) for ProMP. The width of the RBFs are set as $h = (\frac{T-1}{N})^2$. The hyperparameters of the BGMM are the default hyperparameters of the *scikit-learn* library. We choose a diagonal covariance matrix prior, with a standard deviation of 0.1 for the context variables and 1 for the ProMP weights. We use a maximum number of 5 Gaussians, or strictly less than the number of demonstrations if there are less than 6 demonstrations.

Throughout the experiments, we compare our method to three baselines. The first one (**Random**) is a random strategy using the same BGMM representation as our method. The second one (**GP**) is an adaptation of [18] for external context variables: we learn the conditional model of the trajectories given the context with a Gaussian process (GP)[1]

---

[1] Alternatively, we could also learn a GP from contexts to ProMP weights, but in practice it gave the same results as learning directly from contexts to trajectories. For this reason, we do not include it in

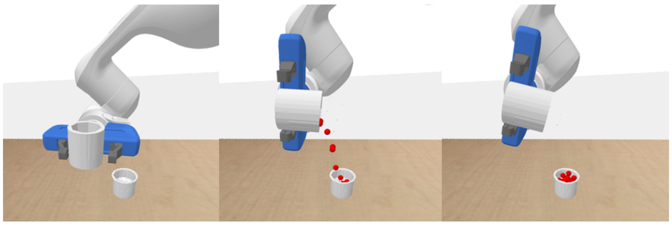


Figure 7.6: Overview of the simulated pouring environment.

using a squared exponential kernel (hyperparameters optimization gave a length scale of 1 and output variance of 0.1^2). The active learning approach for the GP baseline selects the context for which the conditional distribution of the trajectories given the context has the most variance. The third baseline (**Conkey19**) is an adaptation of [19] (introduced in Section 7.1) for external context variables: we learn the joint distribution of contexts and ProMP weights with a GMM and use the Mahalanobis distance in the context space as an active learning measure. We use the same covariance prior as with our approach, and we use $\beta = 3$ for the hyperparameter governing how many outliers are discarded when adding a new datapoint to the Gaussian mixture, see Eq. (7) of [19] for more details[2].

7.4.2 Simulated pouring

We use here a simulated pouring environment implementing the Franka Emika robot in the PyBullet simulator [118]. The goal of this task is to pour liquid (simulated as rigid spherical particles because PyBullet does not support fluids simulation at the time of this work) from a pitcher into a mug. An overview of the simulated setup is shown in Figure 7.6. In the first two simulated environments, we avoid learning the affordances of the object and control directly the orientation of the edge of the pitcher, from where the liquid is poured. This permits us to make the task with a reference trajectory of just one variable: the angle of the pitcher. In the third simulated environment, we go beyond the one-dimensional control angle case, and show the robustness of our approach for more complex movements encoded in a 6-dimensional control variable.

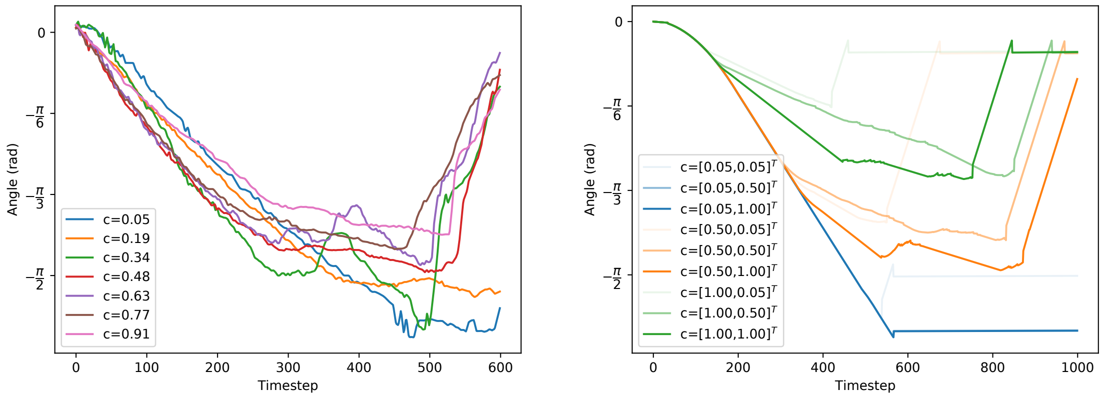
1D context

In this first experiment, we consider a one-dimensional context variable, which represents the amount of liquid in the pitcher. As the mug volume is lower than the pitcher volume, one difficulty of the task is to stop pouring when the mug is full. We consider context variables varying from 0.05 to 1, representing how full the pitcher is (from 5% to 100%). In this experiment, the goal is to fill the mug completely (without overflowing).

---

this work.

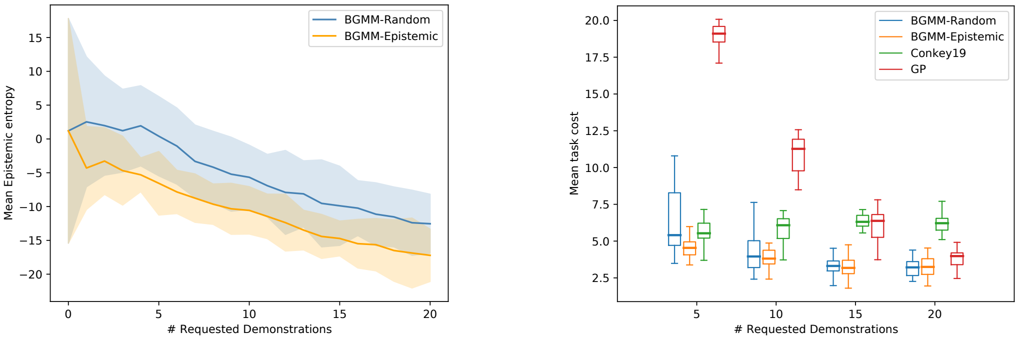
[2] Authors advised to choose β between 2 and 3, we chose 3 because it gave the best results.



(a) Teleoperated demonstrations, 1D context

(b) Generated demonstrations, 2D context

Figure 7.7: Subset of demonstrations for different contexts.



(a) Reduction of epistemic uncertainty w.r.t number of demonstrations

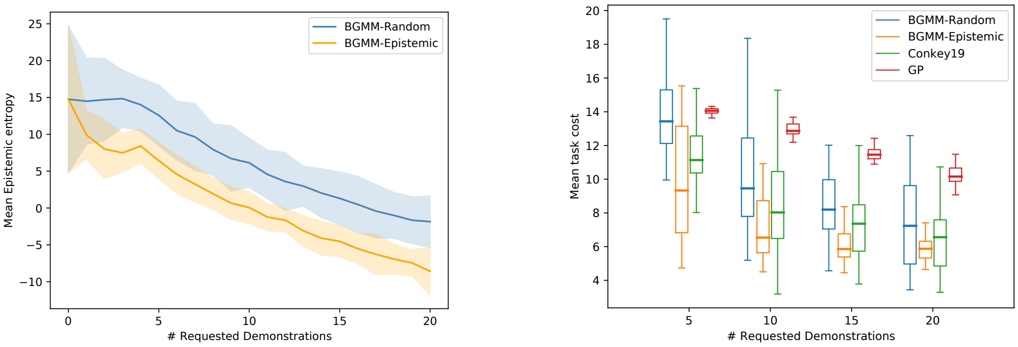
(b) Reduction of task cost w.r.t number of demonstrations

Figure 7.8: Quantitative results for simulated 1D context pouring.

In order to have demonstrations exhibiting realistic variations, we provide real human demonstrations using teleoperation. As the reference trajectory contains only a one-dimensional angle, teleoperation is made simply using a camera by detecting the angle of a colored object held by the human demonstrator. We build a dataset of 100 demonstrations for contexts evenly spread between 0.05 and 1. Namely, we choose $\mathcal{C} = \{0.05 + \frac{1-0.05}{99}k\}_{k=0}^{99}$ and provide one teleoperated demonstration for each context in $\mathcal{C}$. This permits reproducibility of the results and a fair comparison of the methods as they have access to the same demonstrations for given contexts. Demonstrations are aligned using linear interpolation. A subset of aligned demonstrations is shown in Figure 7.7a. We can effectively see that, the more the pitcher is filled, the less it has to be tilted to pour into the mug. We start the active learning process with 2 initial demonstrations, for contexts randomly chosen in the context space $\mathcal{C}$. We make the experiment 20 times with different initial demonstrations. We show in Figure 7.8 how it compares to a random strategy which randomly chooses the next context. In Figure 7.8a, we plot the mean epistemic entropy (averaged on the context space $\mathcal{C}$) in function of the number of requested demonstrations. We can see that our strategy outperforms the random strategy in terms of reduction of the epistemic uncertainties. The diminution of the epistemic uncertainty is particularly big during the first 5 demonstrations requested with our method. In Fig. 7.8b, we propose an objective metric for comparing quantitatively the two methods. We introduce the task cost, which is simply a ℓ_2 norm between the final volume in the mug and the desired final volume (approximated with the number of balls in the mug. The desired number of balls is 50, which corresponds to the mug being almost completely filled. Filling it too much is possible and increases the task cost as well). We observe in Figure 7.8b that our method significantly outperforms the random strategy in the beginning of the learning process (5 demonstrations), while afterwards the results are similar. This suggests that our active learning strategy improves learning with few demonstrations. As the context is low-dimensional (1 dimension), this is not surprising that for more than 10 demonstrations, active learning does not yield any improvement over a random strategy which has also explored the context space well. It is also interesting to note that our method has less variance across experiments than the random strategy. Also, our movement representation with a BGMM gives much better results than the GP approach as it achieves a significantly lower task cost at all stages of the learning process. We can see that our method also outperforms [19], whose performance stagnates during the learning process. We believe this is due to the heuristics that are proposed to add Gaussians to the mixture, which had only been tested in the 2D case in the original paper, and that would probably need to be adjusted.

2D context

In this experiment we propose to add another context variable: the desired final volume in the mug. This context variable also ranges from 0.05 to 1, representing how full the mug is (from 5% to 100%). We then have $c = [c_{\text{pitcher}}, c_{\text{mug}}]^T$. For this task, we manually implement a controller performing the task, which is used as the human demonstrator (note that the demonstrations may not be perfect, e.g., when there is not enough liquid in the pitcher initially to fill the mug to its desired level). A sample of generated demonstrations can be found in Fig. 7.7b. We can see that, for a given desired volume in the mug, the smaller the initial volume of the pitcher is, the more the pitcher needs to be tilted. And, for a given initial volume of the pitcher, the more the mug needs to be filled, the more the object has to be tilted. Note that we do not bring the pitcher back to its horizontal position when it is fully emptied. As in the previous experiment, for reproducibility reasons, we precompute a database of generated demonstrations. A grid of width 20 is used to represent the context space for which demonstrations are generated, yielding 400 demonstrations. Namely, $\mathcal{C} = \{(0.05 + \frac{1-0.05}{99}i, 0.05 + \frac{1-0.05}{99}j)\}_{i,j=0}^{19}$. We also perform 20 experiments where each experiment starts with 2 randomly sampled demonstrations from the database. Results are shown in Figure 7.9. We can see in Figure 7.9a that our strategy outperforms the random strategy in terms of reduction of the epistemic uncertainties. More importantly, we see in Figure 7.9b that the active



(a) Reduction of epistemic uncertainty w.r.t number of demonstrations

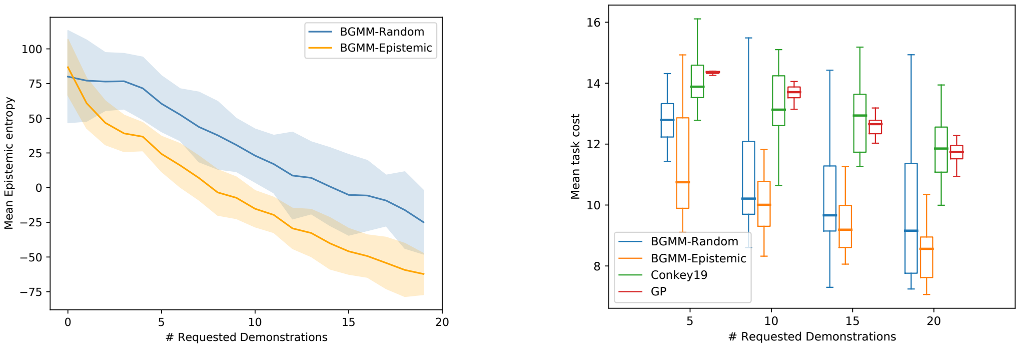
(b) Reduction of task cost w.r.t number of demonstrations

Figure 7.9: Quantitative results for simulated 2D context pouring.

learning strategy can learn the task using fewer demonstrations than a random strategy. Namely, the model improved with 5 demonstrations obtained using our method achieves lower task cost than if the same model was improved with 10 demonstrations using the random strategy. Similarly, 10 actively gathered demonstrations contribute better to the task cost than 20 randomly gathered ones. This shows that the entropy of the epistemic uncertainties of a BGMM is a good metric for actively learning ProMPs. We also observe that our BGMM approach significantly outperforms the GP baseline. In particular, we see that the GP approach is on par with the BGMM-Random approach after 5 requested demonstrations, but then performs worse than the two approaches based on BGMMs. This motivates the use of our Bayesian representation based on ProMPs for learning robot movements, instead of a Gaussian Process approach. Note also that our approach has the additional advantage of quantifying the aleatoric uncertainty as well, which can typically be exploited in ProMPs for designing compliant controllers. Also, we observe that in this experiment the Conkey19 approach performs similarly to our approach, though slightly worse. As explained in the previous subsection, we believe this is because this approach was developed for a 2-dimensional context case.

3D context

In this experiment, we want to test the robustness of our method with respect to higher-dimensional context and control variables. Hence, we add a third context variable related to the position where the pitcher was grasped by the robot. Namely, the robot always starts from the same position but the pitcher can have been grasped at different heights between the base and the top. This makes the movement more complex as one rotation angle is not sufficient anymore to characterize it, and there are correlations between the robot translations and rotations. We use a 6-dimensional control variable consisting of position and orientation (Euler angles) of the robot end-effector. A controller is implemented to execute the task, and is used as the human demonstrator.



(a) Reduction of epistemic uncertainty w.r.t number of demonstrations

(b) Reduction of task cost w.r.t number of demonstrations

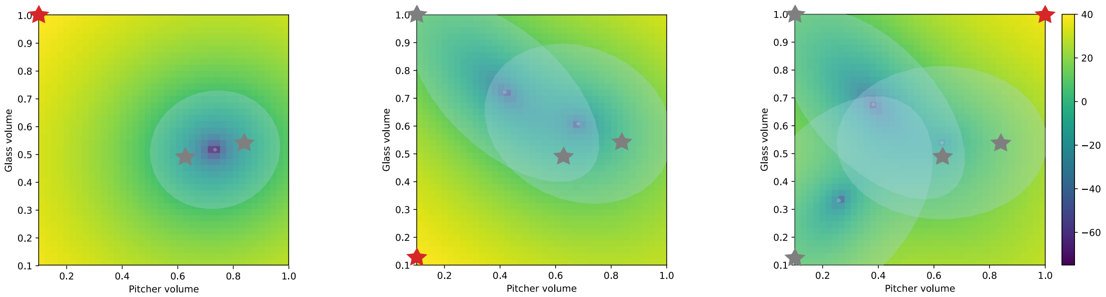
Figure 7.10: Quantitative results for simulated 3D context pouring.

For this experiment, due to the higher dimensionality of the context space, we do not precompute a database of demonstrations as in previous experiments but generate online the demonstrations requested by the algorithm, and use a Bayesian optimization algorithm (the tree-structured Parzen estimator approach [141] implemented in the *hyperopt* Python package [142]) to calculate the context yielding the highest epistemic entropy.

We can see in Figure 7.10a that the reduction of the epistemic uncertainties is bigger with our active learning metric than with the random baseline, similarly to what we observed in the past two experiments, and that this epistemic reduction correlates with a better task cost error (see Figure 7.10b), confirming that the epistemic uncertainties seem to be a good active learning metric. Finally, our method outperforms the two alternative baselines from the literature by a very large margin in this more complicated experiment.

7.4.3 Real robot pouring task

In this experiment we demonstrate the viability of our approach on a pouring task with a real 7-axis Franka Emika robot. An overview of the physical setup can be seen in Figure 7.12. The context space is 2-dimensional as in the previous simulated experiment, with context variables ranging from 10% to 100%. In this experiment, we also show the robustness of our approach to several degrees of freedom as we choose the demonstrations to be 3-dimensional (position in the vertical plane containing the pitcher and the glass, and orientation of the pitcher). We give 2 initial demonstrations to the robot in random contexts, and the robot iteratively requests 20 additional demonstrations. The first 3 iterations of the active learning process are shown in Figure 7.11. We can see that the robot starts by requesting demonstrations at the corners of the state space, which is normal because this is where it is the most uncertain. Note that we could use an



(a) First iteration (b) Second iteration (c) Third iteration

Figure 7.11: Visualization of the context space during the first 3 iterations of the active learning process. The heatmap represents the entropy of the epistemic uncertainty, yellow indicating high uncertainty. Demonstrations are shown as grey stars. The context chosen for the next demonstration is shown as a red star. Transparent ellipses show the marginal distribution of the ProMP in the context space.

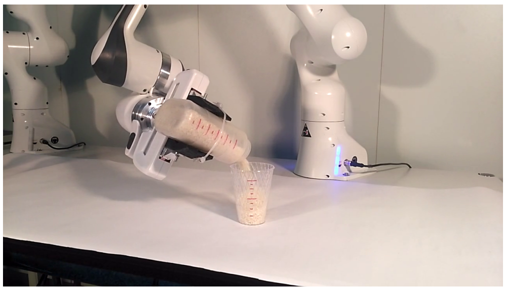


Figure 7.12: Overview of the pouring task with a 7-axis robot.

information-density method to make the requests close to the demonstrations (e.g., by adding a similarity objective). We verified qualitatively that the learned movement representation permits to pour successfully for different contexts, which can be seen on the supplementary video (we tested it on 9 different contexts, taken from a 3×3 grid in the context space).

7.5 Conclusion

This chapter presented a novel active learning framework allowing a robot to ask for informative new demonstrations. The presented framework is based on an information-density cost built from a representation of the epistemic uncertainties of a BGMM model defined at a action-state level and trajectory level. A closed-form cost solution for GMMs can be obtained thanks to the properties of the quadratic Rényi entropy. New query points can then be efficiently obtained by maximizing a GMM approximation of the proposed active learning cost. Our experiments showcase that our approach allows a robot to improve its representation of a task, as well as its corresponding generalization capabilities.

Future work consists in extending our results to theoretically determine a threshold to stop the learning process, which in turn would be useful for determining a sufficient number of demonstrations so that the model can generalize the fastest in the desired space. We believe that the framework can then be used to answer two of the main questions of LfD, which are *i*) Where to give demonstrations? and *i*) How many demonstrations are required?.

8 Discussion & Future Work

In this section, we present a discussion on the limitations and the future work on a selection of topics covered in this thesis. In Section 8.1,

8.1 Projection-based optimization for robotics

In robotics, we use geometric constraints in tasks more often than we realize. In collaborative manipulation tasks such as teleoperation, virtual fixtures or active constraints are used to define a virtual geometric constraint to limit the motion of the robot. This safety measure has been applied as a visual or a haptic constraint defined as geometric entities such as points, hyperplanes, parametric curves, surfaces, or polygonal meshes [143]. In obstacle avoidance with manipulators, a standard way of defining the environment and the robot itself is to use geometric or *collision primitives* as the distance between two such entities can be efficiently computed [144]. When grasping objects, force closures are defined with the help of friction cones [145]. All positive semi-definite matrices (SPD) that we encounter in robotics such as stiffness matrices or manipulability matrices, all define an ellipsoid in space, and the projection of a matrix onto its closest SPD manifold admits an efficient solution (by taking the SVD decomposition and using the box projection to project the singular values to be bigger than 0) [146]. In autonomous driving and autonomous parking, [147], convex 2D polytopes are widely used to model the cars and the obstacles around them.

We believe that the work presented in Chapter 3 promotes the use of projections in constrained optimization problems in robotics, which in turn, hopefully, opens new future directions for faster and cheap-to-compute projections combined with this type of fast solvers. For example, for obstacle avoidance problems, as presented in [144], differentiability of the distances between objects and efficient computation of this derivative is important to achieve real-time tasks. Since the derivative information is either not available or not smooth enough to be used efficiently by the solvers, we see a trend of solving many obstacle avoidance problems with the use of gradient-free sample-based optimizers [148]. Projection-based algorithms, on the other hand, do not require derivative information. The only requirement is a method to compute a manipulator configuration that is closest to a given configuration (which is initially and virtually colliding with an obstacle) and that is not colliding with the obstacle. A future direction would be to find computationally efficient fast projection methods as in [62].

8.2 System level synthesis: perspectives on robust optimal control and inverse optimal control

The system level synthesis (SLS) framework presented in Chapter 4 has been used in distributed control architectures and extended with convex robust optimal control algorithms [85]. The parametrization in SLS allows one to construct convex representations of robust optimal control which would be otherwise not tractable. Robust optimal control formulation when the dynamics model is stochastic is given in [21]. Here, we would like to give future directions on how to apply similar ideas discussed in the robust inverse kinematics experiment in Section 3.5. There, we showed that a hyperplane constraint can be transformed into a second-order cone constraint when the slope of the hyperplane is a random variable following a Gaussian distribution.

We start by letting $\boldsymbol{\Psi_u} = \begin{bmatrix}\boldsymbol{d}_{\boldsymbol{u}}^\top & \boldsymbol{\Phi}_{\boldsymbol{u}}^\top\end{bmatrix}^\top$, $\boldsymbol{\Psi_x} = \begin{bmatrix}\boldsymbol{d}_{\boldsymbol{x}}^\top & \boldsymbol{\Phi}_{\boldsymbol{x}}^\top\end{bmatrix}^\top$ and $\boldsymbol{v} = \begin{bmatrix}1 & \boldsymbol{w}^\top\end{bmatrix}^\top$, such that $\boldsymbol{\Psi_x} = \tilde{\boldsymbol{S}}_{\boldsymbol{x}} + \boldsymbol{S_u}\boldsymbol{\Psi_u}$, where $\tilde{\boldsymbol{S}}_{\boldsymbol{x}} = \begin{bmatrix}\boldsymbol{0}^\top & \boldsymbol{S}_{\boldsymbol{x}}^\top\end{bmatrix}^\top$. In the case of control bounds, we have

$$\begin{aligned}
&\boldsymbol{b}_u \leq \boldsymbol{u} \leq \boldsymbol{c}_u,\\
&\boldsymbol{b}_u \leq \boldsymbol{\Psi_u}\boldsymbol{v} \leq \boldsymbol{c}_u,\\
&\boldsymbol{b}_u^i \leq \boldsymbol{v}^\top\boldsymbol{\Psi}_{\boldsymbol{u}}^i \leq \boldsymbol{c}_u^i, \forall i
\end{aligned}$$

where $\boldsymbol{\Psi}_{\boldsymbol{u}}^i$ is the i^{th} row of $\boldsymbol{\Psi_u}$. This inequality describes the projection of the rows of $\boldsymbol{\Psi_u}$ independently, hence this can be solved analytically in parallel for each $\boldsymbol{\Psi}_{\boldsymbol{u}}^i$. As $\boldsymbol{v}$ is a random variable with $\boldsymbol{v}{\sim}\mathcal{N}(\boldsymbol{\mu_v}, \boldsymbol{\Sigma_v})$, where $\boldsymbol{\mu_v} = [1, \boldsymbol{\mu_w}]$ and $\boldsymbol{\Sigma_v} = \text{blkdiag}(0, \boldsymbol{\Sigma_w})$, we have two options: 1) expected constraints, and 2) robust constraints.

Expected constraints: the inequality constraint becomes

$$\begin{aligned}
&\boldsymbol{b}_u^i \leq \boldsymbol{v}^\top\boldsymbol{\Psi}_{\boldsymbol{u}}^i \leq \boldsymbol{c}_u^i, \forall i, && (8.1)\\
\implies &\boldsymbol{b}_u^i \leq \mathbb{E}_{\boldsymbol{v}}[\boldsymbol{v}^\top\boldsymbol{\Psi}_{\boldsymbol{u}}^i] \leq \boldsymbol{c}_u^i, \forall i, && (8.2)\\
\implies &\boldsymbol{b}_u^i \leq \boldsymbol{\mu}_{\boldsymbol{v}}^\top\boldsymbol{\Psi}_{\boldsymbol{u}}^i \leq \boldsymbol{c}_u^i,\ \forall i, && (8.3)
\end{aligned}$$

The result of the projection depends on $\boldsymbol{\mu_v}$. If we assume standard noise with zero mean, then the result coincides with constrained LQT.

Robust inequalities: We will first solve the upper bound inequality for simplicity, then generalize to the lower bound as well. require that $\mathbb{P}(\boldsymbol{v}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i \leq \boldsymbol{c}_u^i) \geq \eta$ with $\eta \geq 0.5$. We let $z = \boldsymbol{v}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i$ so that $\mu_z = \mathbb{E}_{\boldsymbol{v}}[z] = \boldsymbol{\mu}_{\boldsymbol{v}}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i$ and $\sigma_z^2 = \mathsf{Var}_{\boldsymbol{v}}[z] = {\boldsymbol{\Psi}_{\boldsymbol{u}}^i}^\top \boldsymbol{\Sigma}_{\boldsymbol{v}} \boldsymbol{\Psi}_{\boldsymbol{u}}^i$ and normalize the variable z and rewrite the equation as in

$$
\begin{aligned}
& \mathbb{P}(\boldsymbol{v}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i \leq \boldsymbol{c}_u^i) \geq \eta, \ \ \forall i \in \dim(\boldsymbol{b}^u), \\
\iff & \mathbb{P}(z \leq \boldsymbol{c}_u^i) \geq \eta, \\
\iff & \mathbb{P}(\frac{z - \mu_z}{\sigma_z} \leq \frac{\boldsymbol{c}_u^i - \mu_z}{\sigma_z}) \geq \eta, \\
\iff & \Psi(\frac{\boldsymbol{c}_u^i - \mu_z}{\sigma_z}) \geq \eta, \\
\iff & \frac{\boldsymbol{c}_u^i - \mu_z}{\sigma_z} \geq \Psi^{-1}(\eta), \\
\iff & \mu_z + \Psi^{-1}(\eta)\sigma_z \leq \boldsymbol{c}_u^i, \\
\iff & \boldsymbol{\mu}_{\boldsymbol{v}}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i + \Psi^{-1}(\eta) \|\boldsymbol{\Sigma}_{\boldsymbol{v}}^{1/2} \boldsymbol{\Psi}_{\boldsymbol{u}}^i\|_2 \leq \boldsymbol{c}_u^i,
\end{aligned} \tag{8.4}
$$

where $\Psi(\cdot)$ is the cumulative distribution function of zero mean unit variance Gaussian variable. We remark that this constraint is convex since $\Psi^{-1}(\eta) \geq 0$ as we assumed that $\eta \geq 0.5$. Note that the constraint for the lower bound case can be found with the same method and here we give only the result which also turns out to be convex:

$$
\begin{aligned}
& \mathbb{P}(\boldsymbol{b}_u^i \leq \boldsymbol{v}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i) \geq \eta, \ \ \forall i \in \dim(\boldsymbol{b}_u), \\
\iff & \boldsymbol{b}_u^i \leq \boldsymbol{\mu}_{\boldsymbol{v}}^\top \boldsymbol{\Psi}_{\boldsymbol{u}}^i - \Psi^{-1}(\eta) \|\boldsymbol{\Sigma}_{\boldsymbol{v}}^{1/2} \boldsymbol{\Psi}_{\boldsymbol{u}}^i\|_2.
\end{aligned} \tag{8.5}
$$

The combination of Equation (8.4) and Equation (8.5) implies that each $\boldsymbol{\Psi}_{\boldsymbol{u}}^i$ needs to be projected onto the convex set formed by two second-order cones. In the case of state bounds, we have $\boldsymbol{b}_u \leq \boldsymbol{\Psi}_{\boldsymbol{x}} \boldsymbol{v} \leq \boldsymbol{c}_u$. We can use the same methods described for the control bounds.

As a proof-of-concept, we used the constrained optimization method in [10] as it gives an efficient way of finding feedback controllers in the case of constrained optimization. The method is based on the Alternating Direction Method of Multipliers (ADMM) and can also be considered a projection-based algorithm. We optimized an objective function of reaching a goal position of 0.5 with zero velocity with a 1D double integrator system. The control bounds are between -3 and 3. We solved this problem with LQT-ADMM with non-robust constraints and with SLS-ADMM with robust constraints with a safety guarantee of 80%. We then executed the feedback controllers starting from 1000 different initial positions to test the robustness of the algorithm. Indeed, LQT-ADMM controllers achieve half of the tasks, while SLS-ADMM success rate is around 80%, for the constraint satisfaction. The results are shown in Figure 8.1 for 5 randomly selected initial configurations.

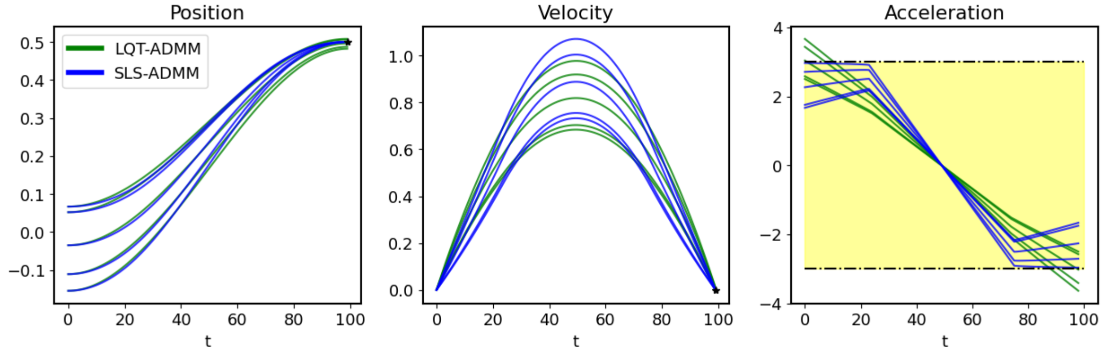


Figure 8.1: Non-robust linear quadratic tracking controller (LQT-ADMM) and robust system level synthesis controller (SLS-ADMM) executed from 5 different initial positions. SLS-ADMM is optimized to be robust with respect to the control bounds.

The SLS-based representation used in our work can facilitate the bridging between learning, planning, and feedback control, especially when we do not know the cost function and/or we do not have prior knowledge on which past states should be correlated with the remaining part of the motion. This provides a very general formulation for robot skill representations. Future work will study which past states are correlated by setting the problem as inverse optimal control. We believe that learning such correlations from demonstrations/experiences will be useful to lead the way to the development of smarter feedback controllers which can act on the memory of the states.

8.3 Combining learning and optimization of controllers

The control methods in robotics have seen their ways merge with the data-driven methods of machine learning thanks to the recent advances in artificial intelligence and computational power. As collecting data in robotics may become quite time-consuming, especially when the dimensionality of the problem increases, learning from demonstration methods have gained popularity. Generalizable movement primitives models are then seen as reference trajectory generators for a stable controller that is designed separately [16]. In [33], Gaussian mixture models (GMMs) are used to define a trajectory model, which is then approximated by a single Gaussian distribution whose mean and precision (inverse of covariance) are used as tracking reference trajectory and tracking precisions. Indeed, such combinations of learning movements and designing controllers have been studied in many works. The disadvantage of such learning methods is that the data gathering and learning processes are completely independent of the controller that is going to be used. In Section 6.2, we showed that for simple tasks with simple dynamics, we can design impedance controllers that can imitate the generalization capabilities of the learned controller. In Section 6.1, we showed how to combine demonstration data in an optimal control problem to warm-start and guide the optimization.

A future direction should be developing learning from demonstration methods that are aware of the controller that they are going to be combined with, if not learning controllers directly. For example, learning feedback controllers in eSLS can be designed as an optimization problem where the decision variable is a distribution of feedback controllers (in the form of tensor-variate Gaussian distributions [149]) that minimizes an objective of matching trajectories found by such a controller to the demonstrated ones. Such an idea has been studied in [47], however, with the drawback that such models are usually not directly generalizable.

Data-driven learning is not restricted to learning motion models. Unmodeled nonlinearities in the dynamics model of collaborative industrial robots renders the control less precise and highly unpredictable [150]. Learning dynamics model of the system provides a way to overcome these problems. Learning the dynamical system from data, however, is not a straightforward problem and there are many solutions proposed in the literature [151, 152, 153, 154, 155]. For example, a nonlinear neural network-based dynamics model would not be suitable for standard and powerful second-order optimal control algorithms such as DDP. A future direction should be to design dynamics models or even improve the existing ones with data-driven approaches such that they are compatible with optimal control. With such a dynamics model of the task at hand, one can design objective functions to achieve many tasks. Such ideas could be exploited in the recent and promising approach to learning Koopman operators [156] from data. Koopman operators are linear operators in high-dimensional states. If one could learn linear dynamics models of a nonlinear model in high dimensional states, it would mean guaranteed stable and efficient control. However, learning such linear dynamics is often very difficult, if not impossible. Yet, there are still promising approximations that are working very well in practice [157, 158, 52]. We believe that the ideas in Koopman operators could be powerful competition to the current state-of-the-art learning methods of dynamics models.

9 Conclusion

In conclusion, this thesis presented a comprehensive study of the challenges in designing adaptive and anticipatory feedback controllers in robot manipulation tasks, by exploring both optimization and learning approaches.

The first part of the thesis focused on optimization methods and aimed to exploit the information contained in the feedback controllers obtained from optimal control. The results showed that exploiting the geometry of the constraints in the optimization process can significantly improve the performance of the solver. Furthermore, the concept of anticipatory memory behavior was introduced via the system level synthesis framework and demonstrated to be a valuable tool for adaptive feedback controllers with changing task parameters. A hierarchical optimal control problem definition was introduced and analytical solutions for linear quadratic cases were obtained.

The second part of the thesis focused on learning from demonstration (LfD) methods, addressing several open questions in the field such as what to model, how to execute the models, how to demonstrate, and how many times to demonstrate. The results showed that the information contained in demonstrations can be leveraged to guide optimal control problems and to design adaptive impedance controllers that mimic the generalization and multimodality capabilities of a learned trajectory policy. We also proposed an active learning framework to iteratively refine trajectory and control policies. We showed that the proposed method reduces the number of demonstrations required to learn a generalizable model.

Overall, the results of this thesis demonstrate the importance of exploiting the structures of the models in both optimization and learning, in order to design adaptive and anticipatory feedback controllers that perform well in real-world scenarios. The new skills discovered, such as anticipatory memory behaviors, adaptability, hierarchy, and robustness, open up exciting avenues for future research and development in the field of robot manipulation.

A Appendix

A.1 Planar Push Dynamics

Table A.1: Constraints of different interaction modes

Sticking	**Sliding Up**	**Sliding Down**	**Separation**
$\{\boldsymbol{v}_f \in \Omega\} \cap \{\boldsymbol{q}_p \in \psi\}$	$\{\Omega < \boldsymbol{v}_f < \phi\} \cap \{\boldsymbol{q}_p \in \psi\}$	$\{\phi < \boldsymbol{v}_f < \Omega\} \cap \{\boldsymbol{q}_p \in \psi\}$	$\{\boldsymbol{v}_f \notin \phi\} \cup \{\boldsymbol{q}_p \notin \psi\}$
$\frac{v_{tf}}{v_{nf}} \leq \gamma_{up},\ \frac{v_{tf}}{v_{nf}} \geq \gamma_{dn},\ v_{nf} > 0,$ $\lvert p_x\rvert = r_s + r_p,\ \lvert p_y\rvert \leq r_s.$	$\frac{v_{tf}}{v_{nf}} > \gamma_{up},\ v_{nf} > 0,$ $\lvert p_x\rvert = r_s + r_p,\quad \lvert p_y\rvert \leq r_s.$	$\frac{v_{tf}}{v_{nf}} < \gamma_{dn},\ v_{nf} > 0,$ $\lvert p_x\rvert = r_s + r_p,\ \lvert p_y\rvert \leq r_s.$	$\vee\, v_{nf} < 0,$ $\vee\, \lvert p_x\rvert > r_s + r_p,$ $\vee\, \lvert p_y\rvert > r_s.$

A.1.1 Kinematics

Fig. A.1 shows the kinematics of the pusher-slider system. The red and blue circles represent the pusher at the initial face and the face after contact switching. The pose of the slider is defined as $\boldsymbol{q}_s = [x\ y\ \theta]^\top$ w.r.t. the global frame $\mathbb{F}_g$, where x and y are the Cartesian coordinates of the center of mass, and θ is the rotation angle around the vertical axis. The contact position between the pusher and the slider is described as $\boldsymbol{q}_p = [p_x\ p_y]^\top$ w.r.t. the current slider frame $\mathbb{F}_f$ after face switching, while $\boldsymbol{q}_f = \boldsymbol{R}_{\theta_f}\boldsymbol{q}_p$ is the expression of $\boldsymbol{q}_p$ in the initial slider frame $\mathbb{F}_s$, and θ_f is the rotation angle from $\mathbb{F}_s$ to $\mathbb{F}_f$. The input of this system is the acceleration of the pusher $\boldsymbol{u} = [\dot{v_n}\ \dot{v_t}]^\top$, which is resolved in $\mathbb{F}_s$, while $\boldsymbol{v} = [v_n\ v_t]^\top$ is the pusher velocity defined in frame $\mathbb{F}_s$, and $\boldsymbol{v}_f = \boldsymbol{R}_{\theta_f}^\top \boldsymbol{v} = [v_{nf}\ v_{tf}]^\top$ is the expression of $\boldsymbol{v}$ in $\mathbb{F}_f$.

A.1.2 Generalized Motion Cone

Based on the quasi-static approximation and ellipsoidal limit surface assumption, a motion cone is introduced to determine the contact mode given by the current pusher velocity and contact position. The two boundaries of the motion cone are given as

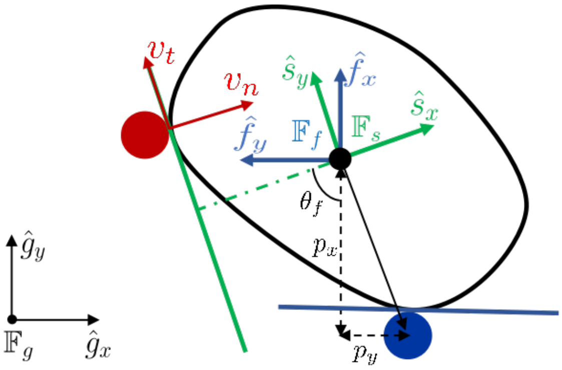


Figure A.1: Kinematics of pusher-slider system allowing face switching.

$\boldsymbol{v_{up}} = 1\boldsymbol{f_x} + \gamma_{up}\boldsymbol{f_y}$ and $\boldsymbol{v_{dn}} = 1\boldsymbol{f_x} + \gamma_{dn}\boldsymbol{f_y}$, resolved in the current slider frame $\mathbb{F}_f$, with

$$\gamma_{up} = \frac{\mu_p c^2 - p_x p_y + \mu_p {p_x}^2}{c^2 + {p_y}^2 - \mu_p p_x p_y}, \tag{A.1}$$

$$\gamma_{dn} = \frac{-\mu_p c^2 - p_x p_y - \mu_p {p_x}^2}{c^2 + {p_y}^2 + \mu_p p_x p_y}, \tag{A.2}$$

where μ_p is the friction coefficient between the pusher and the slider, and c is the parameter connecting applied force and the resulting velocity.

In the previous, the pusher is assumed to remain on one face selected before and not change during the execution. We would like to enable more complex pushing that involves face switching. For that, we introduce another interaction mode, which we call separation mode. Table A.1 lists the relationship between state constraints and interaction modes, where Ω is the set within the boundaries of the motion cone, ϕ is the space where the pusher goes towards the slider, and ψ is the set where the pusher keeps touching with the slider. r_s and r_p are the half length of the slider and the radius of pusher, respectively.

A.1.3 Generalized Motion Equation

We build the generalized motion equation by including separation mode and contact face switching, namely

$$\dot{\boldsymbol{x}} = \begin{cases} \boldsymbol{g}_1(\boldsymbol{x}, \boldsymbol{u}), & \text{if} \quad \textbf{Sticking}, \\ \boldsymbol{g}_2(\boldsymbol{x}, \boldsymbol{u}), & \text{if} \quad \textbf{Sliding Up}, \\ \boldsymbol{g}_3(\boldsymbol{x}, \boldsymbol{u}), & \text{if} \quad \textbf{Sliding Down}, \\ \boldsymbol{g}_4(\boldsymbol{x}, \boldsymbol{u}), & \text{if} \quad \textbf{Separation}, \end{cases} \tag{A.3}$$

where $\boldsymbol{x} = [\boldsymbol{q}_s^\top \ \boldsymbol{q}_f^\top \ \boldsymbol{v}^\top]^\top$, and

$$\boldsymbol{g}_j(\boldsymbol{x},\boldsymbol{u}) = \begin{bmatrix} \begin{bmatrix} \boldsymbol{R}_{\theta_f}\boldsymbol{R}_\theta \boldsymbol{Q}\boldsymbol{P}_j \\ \boldsymbol{b}_j \\ \boldsymbol{R}_{\theta_f}[\boldsymbol{d}_j \quad \boldsymbol{c}_j]^\top \end{bmatrix} \boldsymbol{R}_{\theta_f}^\top \boldsymbol{v} \\ \boldsymbol{u} \end{bmatrix}, \quad \boldsymbol{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

$$\boldsymbol{Q} = \frac{1}{c^2 + p_x^2 + p_y^2} \begin{bmatrix} c^2 + {p_x}^2 & p_x p_y \\ p_x p_y & c^2 + p_y^2 \end{bmatrix},$$

$$\boldsymbol{b}_1 = \begin{bmatrix} \frac{-p_y}{c^2 + p_x^2 + p_y^2} & \frac{p_x}{c^2 + p_x^2 + p_y^2} \end{bmatrix}, \boldsymbol{b}_2 = \begin{bmatrix} \frac{-p_y + \gamma_{up} p_x}{c^2 + p_x^2 + p_y^2} & 0 \end{bmatrix},$$

$$\boldsymbol{b}_3 = \begin{bmatrix} \frac{-p_y + \gamma_{dn} p_x}{c^2 + p_x^2 + p_y^2} & 0 \end{bmatrix}, \quad \boldsymbol{b}_4 = [0 \quad 0],$$

$$\boldsymbol{c}_1 = [0 \quad 0], \quad \boldsymbol{c}_2 = [-\gamma_{up} \quad 1], \boldsymbol{c}_3 = [-\gamma_{dn} \quad 1], \boldsymbol{c}_4 = [0 \quad 1],$$

$$\boldsymbol{d}_1 = [0 \quad 0], \quad \boldsymbol{d}_2 = [0 \quad 0], \quad \boldsymbol{d}_3 = [0 \quad 0], \quad \boldsymbol{d}_4 = [1 \quad 0],$$

$$\boldsymbol{P}_1 = \boldsymbol{I}_{2\times 2}, \quad \boldsymbol{P}_2 = \begin{bmatrix} 1 & 0 \\ \gamma_{up} & 0 \end{bmatrix}, \boldsymbol{P}_3 = \begin{bmatrix} 1 & 0 \\ \gamma_{dn} & 0 \end{bmatrix}, \quad \boldsymbol{P}_4 = \boldsymbol{0}_{2\times 2}.$$

where $j = 1, 2, 3, 4$ corresponds to sticking, sliding up, sliding down, and separation mode, respectively.

A.2 Mathematical background

A.2.1 Pseudo-inverse and nullspace of a row matrix

Let $\boldsymbol{J} \in \mathbb{R}^{m\times n}$ be a matrix with rank r, i.e., $\text{rank}(\boldsymbol{J}) = r \leq \min(m,n)$. Let $\boldsymbol{J}{=}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the SVD decomposition of $\boldsymbol{J}$ which can be expressed with partial matrices separated according to the zero singular values as

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^\top \\ \boldsymbol{V}_2^\top \end{bmatrix}, \tag{A.4}$$

for any $\boldsymbol{J} \in \mathbb{R}^{m\times n}$. Here $\boldsymbol{U}_1 \in \mathbb{R}^{m\times r}$, $\boldsymbol{U}_2 \in \mathbb{R}^{m\times m-r}$, $\boldsymbol{V}_1^\top \in \mathbb{R}^{r\times n}$, $\boldsymbol{V}_2^\top \in \mathbb{R}^{n-r\times n}$ and $\boldsymbol{\Sigma}_1^\top \in \mathbb{R}^{r\times r}$, where $\boldsymbol{\Sigma}_1$ is the diagonal matrix of non-zero singular values of $\boldsymbol{J}$. Then we can define the pseudoinverse and the nullspace matrix as

$$\boldsymbol{J}^\dagger = \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^{-1} \boldsymbol{U}_1^\top, \tag{A.5}$$

$$\boldsymbol{N} = \boldsymbol{V}_2 \boldsymbol{V}_2^\top. \tag{A.6}$$

Using these notations, one can notice that the only time the nullspace matrix does not exist, is when $r = n$ since $\boldsymbol{V}_2$ does not exist (see the dimensions). The right inverse and the left inverse of a matrix are special cases of full row rank and full column rank, respectively. Instead, we use the general SVD decomposition procedure to compute pseudoinverses and nullspace matrices.

A regularized pseudoinverse matrix and the corresponding nullspace matrix can be written as

$$\boldsymbol{J}^{\ddagger} = \boldsymbol{V}_1 \boldsymbol{\Sigma}_1^{\ddagger} \boldsymbol{U}_1^{\top}, \tag{A.7}$$

$$\boldsymbol{N} = \boldsymbol{V}_2 \boldsymbol{V}_2^{\top} + \boldsymbol{V}_1 (\boldsymbol{I} - \boldsymbol{\Sigma}_1^{\ddagger} \boldsymbol{\Sigma}_1) \boldsymbol{V}_1^{\top}, \tag{A.8}$$

where $\boldsymbol{\Sigma}_1^{\ddagger} = \boldsymbol{\Sigma}_1 (\boldsymbol{\Sigma}_1^2 + \lambda \boldsymbol{I})^{-1}$. Notice that in the case of $r = n$, the nullspace (that did not exist before) has now some value because even though $\boldsymbol{V}_2$ does not exist, the second term in (A.8) is not null and can be denoted as an error term in the calculation of nullspace matrix when we use regularization.

A useful property to note is

$$(\boldsymbol{J}^{\top} \boldsymbol{J})^{\dagger} \boldsymbol{J}^{\top} = \boldsymbol{J}^{\dagger}. \tag{A.9}$$

which can be proved easily using the SVD decomposition of $\boldsymbol{J}$.

A.2.2 Properties of the nullspace matrix

A nullspace matrix $\boldsymbol{N}$ is an orthogonal projection matrix with the following properties:

1. $\boldsymbol{N}$ is a symmetric and idempotent matrix with $\boldsymbol{N} = \boldsymbol{N}^2 = \boldsymbol{N}^{\top} = \boldsymbol{N}^{\dagger}$
2. Its eigenvalues are either 0 and 1, showing that it is also a positive semi-definite matrix.
3. For any matrix $\boldsymbol{B}$, it satisfies $\boldsymbol{N}(\boldsymbol{B}\boldsymbol{N})^{\dagger} = (\boldsymbol{B}\boldsymbol{N})^{\dagger}$ and $(\boldsymbol{N}\boldsymbol{B})^{\dagger}\boldsymbol{N} = (\boldsymbol{N}\boldsymbol{B})^{\dagger}$.
4. $\boldsymbol{N}(\boldsymbol{A}) = \boldsymbol{N}(\boldsymbol{A}^{\top}\boldsymbol{A})$.

A.2.3 Separable costs and constraints

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a lower triangular matrix, $\boldsymbol{M} \in \mathbb{R}^{m \times m}$ be a matrix and $\boldsymbol{N} \in \mathbb{R}^{n \times n}$ be a diagonal positive definite matrix. Let the superscript i of a matrix denote its column vector (e.g. the vector $\boldsymbol{A}^i \in \mathbb{R}^m$ is the i^{th} column of $\boldsymbol{A}$) and the superscript j of a matrix denote its row vector (e.g. the vector $\boldsymbol{A}^j \in \mathbb{R}^n$ is the j^{th} row of $\boldsymbol{A}$). We first observe that $(\boldsymbol{A}\boldsymbol{N})^i{=}\boldsymbol{A}\boldsymbol{N}^i{=}\boldsymbol{A}^i\boldsymbol{N}^{ii}$ as $\boldsymbol{N}$ is a diagonal matrix, and $(\boldsymbol{M}\boldsymbol{A})^i{=}\boldsymbol{M}\boldsymbol{A}^i{=}\boldsymbol{M}^{i:}\boldsymbol{A}^i$ as $\boldsymbol{A}$ is a lower triangular matrix, where $\boldsymbol{M}^{i:}$ is the submatrix obtained by removing all the rows and columns before the index i. Then, we can obtain the following identities:

$$\begin{aligned}
\|\boldsymbol{A}\|_F^2 &= \sum_i \sum_j A_{ij} = \sum_i \|\boldsymbol{A}^i\|_2^2 = \sum_j \|\boldsymbol{A}^j\|_2^2, && \text{(A.10)}\\
\|\boldsymbol{AN}\|_F^2 &= \sum_i \|(\boldsymbol{AN})^i\|_2^2 = \sum_i \|\boldsymbol{A}^i \boldsymbol{N}^{ii}\|_2^2, && \text{(A.11)}\\
\|\boldsymbol{MA}\|_F^2 &= \sum_i \|(\boldsymbol{MA})^i\|_2^2 = \sum_i \|\boldsymbol{M}^{i:} \boldsymbol{A}^i\|_2^2, && \text{(A.12)}\\
\iff \|\boldsymbol{MAN}\|_F^2 &= \sum_i \|\boldsymbol{M}^{i:} \boldsymbol{A}^i \boldsymbol{N}^{ii}\|_2^2 && \text{(A.13)}
\end{aligned}$$

Let $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{R} \in \mathbb{R}^{n\times m}$ be lower triangular matrices. Then, the equality $\boldsymbol{B}{=}\boldsymbol{P} + \boldsymbol{RA}$ can be column-wise separated as $\boldsymbol{B}^i{=}\boldsymbol{P}^i + \boldsymbol{R}^{i:}\boldsymbol{A}^i$.

A.2.4 Expectation of some linear and quadratic forms

Let $\boldsymbol{x}{\sim}\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{Ax} + \boldsymbol{a}] &= \boldsymbol{A\mu} + \boldsymbol{a}, && \text{(A.14)}\\
\mathbb{E}_{\boldsymbol{x}}[\|\boldsymbol{Ax} + \boldsymbol{a}\|_2^2] &= \|\boldsymbol{A\Sigma}^{\frac{1}{2}}\|_F^2 + \|\boldsymbol{a}\|_2^2, && \text{(A.15)}\\
\mathbb{E}_{\boldsymbol{x}}[\|\boldsymbol{Ax} + \boldsymbol{a}\|_{\boldsymbol{Q}}^2] &= \|\boldsymbol{A\Sigma}^{\frac{1}{2}}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{a}\|_{\boldsymbol{Q}}^2, && \text{(A.16)}\\
\mathbb{E}_{\boldsymbol{x}}[(\boldsymbol{Ax} + \boldsymbol{a})^\top(\boldsymbol{Bx} + \boldsymbol{b})] &= \mathrm{Tr}[\boldsymbol{A\Sigma B}] + \boldsymbol{a}^\top \boldsymbol{b} && \text{(A.17)}
\end{aligned}$$

A.2.5 Probabilistic inequalities

Assuming $z \sim \mathcal{N}(\mu_z, \sigma_z)$, if we would like to achieve the inequality $z \le c$ with a probability of $\eta \ge 0.5$, we write

$$\begin{aligned}
&\mathbb{P}(z \le c) \ge \eta,\\
\iff &\mathbb{P}(\frac{z-\mu_z}{\sigma_z} \le \frac{c-\mu_z}{\sigma_z}) \ge \eta,\\
\iff &\Psi(\frac{c-\mu_z}{\sigma_z}) \ge \eta,\\
\iff &\frac{c-\mu_z}{\sigma_z} \ge \Psi^{-1}(\eta),\\
\iff &\mu_z + \Psi^{-1}(\eta)\sigma_z \le c, && \text{(A.18)}
\end{aligned}$$

where $\Psi(\cdot)$ is the cumulative distribution function of zero mean unit variance Gaussian variable. Assuming $z = \boldsymbol{a}^\top \boldsymbol{x}$, where $\boldsymbol{x}$ is the decision variable of an optimization problem and $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, we can write $\mu_z = \boldsymbol{\mu}^\top \boldsymbol{x}$ and $\sigma_z = \|\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{x}\|_2$. The constraint of achieving the inequality $\boldsymbol{a}^\top \boldsymbol{x} \le c$ with a probability of $\eta \ge 0.5$ can be expressed as $\mathbb{P}(\boldsymbol{a}^\top \boldsymbol{x} \le c) \ge \eta \iff \boldsymbol{\mu}^\top \boldsymbol{x} + \Psi^{-1}(\eta)\|\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{x}\|_2 \le c$, which is a second-order cone constraint.

A.2.6 Projection onto sublevel set of a convex function

Theorem: Solutions to the optimization problems in the form $\min_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{z}\| \quad \text{s.t.} \quad f(\boldsymbol{x}) \leq t$ is called a *projection* onto the sublevel set of a convex function and denoted as $\Pi_{\mathcal{C}}(\boldsymbol{z})$ with the convex set $\mathcal{C} = \{\boldsymbol{x} | f(\boldsymbol{x}) - t \leq 0\}$. This projection is given by $\Pi_{\mathcal{C}}(\boldsymbol{z}) = (\boldsymbol{I} + \mu \boldsymbol{\partial} f)^{-1}(\boldsymbol{z})$, where μ is an arbitrary solution of $f(\Pi_{\mathcal{C}}(\boldsymbol{z})) = t$.

Affine projections: If we let $f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x}$ with $\mathcal{C} = \{\boldsymbol{x} | \boldsymbol{a}^\top \boldsymbol{x} \leq u\}$, then $\boldsymbol{\partial} f = \boldsymbol{a}$, hence

$$\begin{aligned} (\boldsymbol{I} + \mu \boldsymbol{\partial} f)(\Pi_{\mathcal{C}}(\boldsymbol{z})) &= \boldsymbol{z}, \\ \Pi_{\mathcal{C}}(\boldsymbol{z}) + \mu \boldsymbol{a} &= \boldsymbol{z}, \\ \Pi_{\mathcal{C}}(\boldsymbol{z}) &= \boldsymbol{z} - \mu \boldsymbol{a} \end{aligned}$$

Then, we find an arbitrary solution for μ from $f(\Pi_{\mathcal{C}}(\boldsymbol{z})) = u$ as

$$\begin{aligned} f(\boldsymbol{z} - \mu \boldsymbol{a}) &= u, \\ \boldsymbol{a}^\top (\boldsymbol{z} - \mu \boldsymbol{a}) &= u, \\ \mu &= (\boldsymbol{a}^\top \boldsymbol{z} - u) / \|\boldsymbol{a}\|_2^2. \end{aligned}$$

One can find the projection onto $\mathcal{C} = \{\boldsymbol{x} | l \leq \boldsymbol{a}^\top \boldsymbol{x}\}$ by letting $f(\boldsymbol{x}) = -\boldsymbol{a}^\top \boldsymbol{x}$ and replacing u by $-l$.

Note that if we take $\boldsymbol{x}$ to be one-dimensional and $a=1$, then the problem becomes the projection onto bounds $\mathcal{C} = \{x | x \leq u\}$ which can be solved with the "clipping" operator defined by $\Pi_{\mathcal{C}}(z) = \min(z, u)$. This can be extended to lower bounds with $\Pi_{\mathcal{C}}(z) = \max(\min(z, u), l)$.

Quadratic projections: If we let $f(\boldsymbol{x}) = \frac{1}{2} \boldsymbol{x}^\top \boldsymbol{x}$ with $\mathcal{C} = \{\boldsymbol{x} | \frac{1}{2} \boldsymbol{x}^\top \boldsymbol{x} \leq u\}$, then, $\boldsymbol{\partial} f = \boldsymbol{x}$, hence

$$\begin{aligned} (\boldsymbol{I} + \mu \boldsymbol{\partial} f)(\Pi_C(\boldsymbol{z})) &= \boldsymbol{z}, \\ \Pi_C(\boldsymbol{z}) + \mu P_C(\boldsymbol{z}) &= \boldsymbol{z}, \\ \Pi_C(\boldsymbol{z}) &= (1 + \mu)^{-1} \boldsymbol{z} \end{aligned}$$

Then, we find an arbitrary solution for μ from $f(\Pi_C(\boldsymbol{z}))=u$ from

$$\begin{aligned} f((1+\mu)^{-1}\boldsymbol{z}) &= u, \\ \frac{1}{2}(1+\mu)^{-2}\boldsymbol{z}^\top\boldsymbol{z} &= u, \\ (1+\mu)^2 &= \frac{\|\boldsymbol{z}\|^2}{2u}, \\ \mu &= -1 \pm \frac{\|\boldsymbol{z}\|}{\sqrt{2u}}. \end{aligned}$$

One can find the projection onto $\mathcal{C}=\{\boldsymbol{x}|l\leq\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x}\}$ by letting $f(\boldsymbol{x})=-\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{x}$ and replacing u with $-l$.

Second-order cone projections: Let $\mathcal{C}$ be the standard second-order cone $\mathcal{C}=\{(\boldsymbol{z},t)\in\mathcal{Z}\times\mathbb{R}|\|\boldsymbol{z}\|_2\leq t\}$. Then the projection of a point $(\boldsymbol{z},t)$ onto $\mathcal{C}$ is given by

$$\Pi_{\mathcal{C}}(\boldsymbol{z},t) = \begin{cases} (\boldsymbol{z},t), & \text{if } \|\boldsymbol{z}\|\leq t \\ (\boldsymbol{0},0), & \text{if } \|\boldsymbol{z}\|\leq -t \\ \frac{\|\boldsymbol{z}\|+t}{2}(\frac{\boldsymbol{z}}{\|\boldsymbol{z}\|},1) & \text{otherwise} \end{cases}$$

Square projections: If we let $f(\boldsymbol{x})=\|\boldsymbol{x}\|_\infty$ with $\mathcal{C}=\{\boldsymbol{x}|\|\boldsymbol{x}\|_\infty\leq u\}$, then $\mathcal{C}$ defines a square region centered at $\boldsymbol{0}$ with side length of $2u$.

A.3 Controller derivation

A.3.1 eSLS closed-loop map

For $\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{\Sigma}_w)$, where Σ_w is a diagonal matrix with diagonal entries denoted as σ^i, taking the expectation of the SLS cost defined in (4.6), we obtain

$$\begin{aligned} J_{\text{SLS}} &= \mathbb{E}_{\boldsymbol{w}}[\|\boldsymbol{\Phi}_x\boldsymbol{w}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{\Phi}_u\boldsymbol{w}\|_{\boldsymbol{R}}^2], \\ &= \|\boldsymbol{\Phi}_x\boldsymbol{\Sigma}_w^{\frac{1}{2}}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{\Phi}_u\boldsymbol{\Sigma}_w^{\frac{1}{2}}\|_{\boldsymbol{R}}^2, && \text{(A.19)} \\ &= \sum_{i=1}^{T} \underbrace{\|\boldsymbol{\Phi}_x^i\boldsymbol{\sigma}_w^i\|_{\boldsymbol{Q}^{i:}}^2 + \|\boldsymbol{\Phi}_u^i\boldsymbol{\sigma}_w^i\|_{\boldsymbol{R}^{i:}}^2}_{J_{\text{SLS}}^i(\boldsymbol{\Phi}_{\boldsymbol{x}}^i,\boldsymbol{\Phi}_{\boldsymbol{u}}^i)}, && \text{(A.20)} \end{aligned}$$

which shows that the cost function can be separated into T terms each (i) depending on only on the i^{th} block-column of $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_u$. We can separate column-wise also the dynamics constraint $\boldsymbol{\Phi}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{x}} + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{\Phi}_{\boldsymbol{u}}$ as in $\boldsymbol{\Phi}_{\boldsymbol{x}}^i = \boldsymbol{S}_{\boldsymbol{x}}^i + \boldsymbol{S}_{\boldsymbol{u}}^{i:}\boldsymbol{\Phi}_{\boldsymbol{u}}^i$ and set up the SLS optimization subproblem i as

$$\begin{aligned} \min_{\boldsymbol{\Phi}^i_{\boldsymbol{x}}, \boldsymbol{\Phi}^i_{\boldsymbol{u}}} \quad & \textstyle\sum_{i=1}^{T} J^i_{\mathrm{SLS}}(\boldsymbol{\Phi}^i_{\boldsymbol{x}}, \boldsymbol{\Phi}^i_{\boldsymbol{u}}) \\ \text{s.t.} \quad & \boldsymbol{\Phi}^i_{\boldsymbol{x}} = \boldsymbol{S}^i_{\boldsymbol{x}} + \boldsymbol{S}^{i:}_{\boldsymbol{u}} \boldsymbol{\Phi}^i_{\boldsymbol{u}}, \end{aligned} \tag{A.21}$$

The solution to (A.21) for each $i \in [1, \cdots, T]$ can be found analytically. By inserting the linear constraint inside the cost function, we convert (A.21) to a least-square problem by $\boldsymbol{\Phi}^{i*}_{\boldsymbol{u}} = \operatorname{arg\,min}_{\boldsymbol{\Phi}^i_{\boldsymbol{u}}} J^i_{\mathrm{SLS}}(\boldsymbol{\Phi}^i_{\boldsymbol{u}}) = \|(\boldsymbol{S}^i_{\boldsymbol{x}} + \boldsymbol{S}^{i:}_{\boldsymbol{u}} \boldsymbol{\Phi}^i_{\boldsymbol{u}}) \boldsymbol{\sigma}^i_w\|^2_{\boldsymbol{Q}^{i:}} + \|\Phi^i_u \boldsymbol{\sigma}^i_w\|^2_{\boldsymbol{R}^{i:}}$ which admits the following analytical solution:

$$\boldsymbol{\Phi}^{i*}_{\boldsymbol{u}} = -(\boldsymbol{S}^{i:\top}_{\boldsymbol{u}} \boldsymbol{Q}^{i:} \boldsymbol{S}^{i:}_{\boldsymbol{u}} + \boldsymbol{R}^{i:})^{-1} \boldsymbol{S}^{i:\top}_{\boldsymbol{u}} \boldsymbol{Q}^{i:} \boldsymbol{S}^i_{\boldsymbol{x}}, \tag{A.22}$$

$$\boldsymbol{\Phi}^{i*}_{\boldsymbol{x}} = \boldsymbol{S}^i_{\boldsymbol{x}} + \boldsymbol{S}^{i:}_{\boldsymbol{u}} \boldsymbol{\Phi}^{i*}_{\boldsymbol{u}}. \tag{A.23}$$

The eSLS cost can be expressed as

$$\begin{aligned} J_{\mathrm{eSLS}} &= \mathbb{E}_{\boldsymbol{w}}[\|\boldsymbol{\Phi}_{\boldsymbol{x}} \boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{x}} - \boldsymbol{\mu}_{\boldsymbol{x}}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{\Phi}_{\boldsymbol{u}} \boldsymbol{w} + \boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\mu}_{\boldsymbol{u}}\|^2_{\boldsymbol{R}}], \\ &= \|\boldsymbol{\Phi}_x \boldsymbol{\Sigma}^{\frac{1}{2}}_w\|^2_{\boldsymbol{Q}} + \|\boldsymbol{d}_{\boldsymbol{x}} - \boldsymbol{\mu}_{\boldsymbol{x}}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{\Phi}_u \boldsymbol{\Sigma}^{\frac{1}{2}}_w\|^2_{\boldsymbol{R}} + \|\boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\mu}_{\boldsymbol{u}}\|^2_{\boldsymbol{R}}, \\ &= J(\boldsymbol{d}_{\boldsymbol{x}}, \boldsymbol{d}_{\boldsymbol{u}}) + \sum_{i=1}^{T} J^i_{\mathrm{SLS}}(\boldsymbol{\Phi}^i_{\boldsymbol{x}}, \boldsymbol{\Phi}^i_{\boldsymbol{u}}) \end{aligned} \tag{A.24}$$

which shows that the cost function can be decomposed into a part $J(\boldsymbol{d}_{\boldsymbol{x}}, \boldsymbol{d}_{\boldsymbol{u}}) = \|\boldsymbol{d}_{\boldsymbol{x}} - \boldsymbol{\mu}_{\boldsymbol{x}}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\mu}_{\boldsymbol{u}}\|^2_{\boldsymbol{R}}$ that optimizes separately the feedforward pair $\{\boldsymbol{d}_{\boldsymbol{x}}, \boldsymbol{d}_{\boldsymbol{u}}\}$ and a part $J^i_{\mathrm{SLS}}(\boldsymbol{\Phi}^i_{\boldsymbol{x}}, \boldsymbol{\Phi}^i_{\boldsymbol{u}})$ that optimizes the feedback pair $\{\boldsymbol{\Phi}^i_{\boldsymbol{x}}, \boldsymbol{\Phi}^i_{\boldsymbol{u}}\}$. The solution to the feedback pair is given by (A.23) and the solution to the feedforward pair can be determined solving the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{d}_{\boldsymbol{x}}, \boldsymbol{d}_{\boldsymbol{u}}} \quad & J(\boldsymbol{d}_{\boldsymbol{x}}, \boldsymbol{d}_{\boldsymbol{u}}) \\ \text{s.t.} \quad & \boldsymbol{d}_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{d}_{\boldsymbol{u}} \end{aligned} \tag{A.25}$$

The solution to (A.25) can be found analytically. By inserting the linear equality constraint inside the cost function, we convert (A.25) to a least-square problem by $\boldsymbol{d}^*_{\boldsymbol{u}} = \operatorname{arg\,min}_{\boldsymbol{d}_{\boldsymbol{u}}} \|\boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\mu}_{\boldsymbol{x}}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{d}_{\boldsymbol{u}} - \boldsymbol{\mu}_{\boldsymbol{u}}\|^2_{\boldsymbol{R}}$ which admits the following analytical solution:

$$\boldsymbol{d}^*_{\boldsymbol{u}} = (\boldsymbol{S}^{\top}_{\boldsymbol{u}} \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R})^{-1} (\boldsymbol{S}^{\top}_{\boldsymbol{u}} \boldsymbol{Q} \boldsymbol{\mu}_{\boldsymbol{x}} + \boldsymbol{R} \boldsymbol{\mu}_{\boldsymbol{u}}), \tag{A.26}$$

$$\boldsymbol{d}^*_{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{d}^*_{\boldsymbol{u}} \tag{A.27}$$

A.4 Nullspace in inverse kinematics and optimal control

A.4.1 Nullspace method for linear equality constrained QP

A linear equality constrained quadratic program is defined as

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \|\boldsymbol{A}_2\boldsymbol{q} - \boldsymbol{b}_2\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{A}_1\boldsymbol{q} - \boldsymbol{b}_1 = 0, \end{aligned} \tag{A.28}$$

which can be solved by nullspace methods assuming $\boldsymbol{A}_i \in \mathbb{R}^{m\times n}$ are matrices with rank r such that $r \leq \min(m < n))$ as follows. First, the solution to the linear constraint $\boldsymbol{A}_1\boldsymbol{q} - \boldsymbol{b}_1 = 0$ can be written as $\boldsymbol{q} = \boldsymbol{A}_1^\dagger\boldsymbol{b}_1 + \boldsymbol{N}_1\boldsymbol{y}$, with an arbitrary $\boldsymbol{y}$. Plugging this into *Equation* (A.28) results in the following unconstrained optimization problem in $\boldsymbol{y}$:

$$\min_{\boldsymbol{y}} \quad \|\boldsymbol{A}_2(\boldsymbol{A}_1^\dagger\boldsymbol{b}_1 + \boldsymbol{N}_1\boldsymbol{y}) - \boldsymbol{b}_2\|_2^2 \tag{A.29}$$

which admits the analytical solution

$$\boldsymbol{y}^* = (\boldsymbol{A}_2\boldsymbol{N}_1)^\dagger(\boldsymbol{b}_2 - \boldsymbol{A}_2\boldsymbol{A}_1^\dagger\boldsymbol{b}_1), \tag{A.30}$$

A.4.2 Bilevel optimization

A bilevel optimization problem in its simplest form is described as

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \tfrac{1}{2}\|\boldsymbol{f}_2(\boldsymbol{q})\|_2^2 \\ \text{s.t.} \quad & \tfrac{1}{2}\|\boldsymbol{f}_1(\boldsymbol{q})\|_2^2 \quad \text{is minimum,} \end{aligned} \tag{A.31}$$

The optimality conditions (can be extended to KKT condition in its constrained versions) for the lower level problem are $\boldsymbol{J}_1(\boldsymbol{q})^\top\boldsymbol{f}_1(\boldsymbol{q}) = \boldsymbol{0}$, where $\boldsymbol{J}_1(\boldsymbol{q}) = \nabla_{\boldsymbol{u}}\boldsymbol{f}_1(\boldsymbol{q}) \in \mathbb{R}^{m\times n}$ are matrices with rank r such that $r \leq \min(m < n)$. Note that in this setting, the only time we do not have a nullspace matrix is when $r = n$. Using the optimality conditions, we transform (A.31) into a single level optimization problem as

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \tfrac{1}{2}\|\boldsymbol{f}_2(\boldsymbol{q})\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{J}_1(\boldsymbol{q})^\top\boldsymbol{f}_1(\boldsymbol{q}) = \boldsymbol{0} \end{aligned} \tag{A.32}$$

which can be solved by constrained Gauss-Newton method. First, we linearize the constraints around a point $\boldsymbol{q}_k$ as

$$\boldsymbol{J}_1(\boldsymbol{q})^\top\boldsymbol{f}_1(\boldsymbol{q}) \approx \boldsymbol{J}_1(\boldsymbol{q}_k)^\top\boldsymbol{f}_1(\boldsymbol{q}_k) + \boldsymbol{J}_1(\boldsymbol{q}_k)^\top\boldsymbol{J}_1(\boldsymbol{q}_k)\delta\boldsymbol{q} = 0, \tag{A.33}$$

$$\implies \delta\boldsymbol{q} = -\Big(\boldsymbol{J}_1(\boldsymbol{q}_k)^\top\boldsymbol{J}_1(\boldsymbol{q}_k)\Big)^{-1}\boldsymbol{J}_1(\boldsymbol{q}_k)^\top\boldsymbol{f}_1(\boldsymbol{q}_k) + \boldsymbol{N}_1(\boldsymbol{q}_k)\boldsymbol{y}, \tag{A.34}$$

$$\implies \delta\boldsymbol{q} = -\boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger\boldsymbol{f}_1(\boldsymbol{q}_k) + \boldsymbol{N}_1(\boldsymbol{q}_k)\boldsymbol{y},, \tag{A.35}$$

where $\boldsymbol{y}$ is arbitrary, and the last equality is found by using the pseudoinverse property Equation (A.9) and the nullspace property in Appendix A.2.2.

Linearizing the objective function, we obtain the following subproblem

$$\min_{\boldsymbol{y}} \quad \|\boldsymbol{J}_2(\boldsymbol{q}_k)\Big(-\boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger \boldsymbol{f}_1(\boldsymbol{q}_k) + \boldsymbol{N}_1(\boldsymbol{q}_k)\boldsymbol{y}\Big) + \boldsymbol{f}_2(\boldsymbol{q}_k)\|_2^2 \tag{A.36}$$

which is a linearly constrained QP that can be solved by the method in Appendix A.4.1. The solution can be directly obtained as

$$\boldsymbol{y}^* = \Big(\boldsymbol{J}_2(\boldsymbol{q}_k)\boldsymbol{N}_1(\boldsymbol{q}_k)\Big)^\dagger \Big(-\boldsymbol{f}_2(\boldsymbol{q}_k) + \boldsymbol{J}_2(\boldsymbol{q}_k)\boldsymbol{J}_1(\boldsymbol{q}_k)^\dagger \boldsymbol{f}_1(\boldsymbol{q}_k)\Big) \tag{A.37}$$

Bibliography

[1] G. Antonelli, “Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems,” *IEEE Transactions on Robotics*, vol. 25, no. 5, pp. 985–994, Oct 2009.

[2] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems.” in *International Conference on Informatics in Control, Automation and Robotics*, 2004, pp. 222–229.

[3] D. Mayne, “A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems,” *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[4] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, “High-frequency nonlinear model predictive control of a manipulator,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2021, pp. 7330–7336.

[5] P. E. Gill, W. Murray, and M. A. Saunders, “Snopt: An sqp algorithm for large-scale constrained optimization.” *SIAM J. Optimization*, vol. 12, no. 4, pp. 979–1006, 2002.

[6] D. Kraft, “A software package for sequential quadratic programming,” *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Heidelberg, Berlin, New-York: Springer-Verlag, 1992.

[8] A. Wachter, “An interior point algorithm for large-scale nonlinear optimization with applications in process engineering,” Ph.D. dissertation, Carnegie Mellon University, 2002.

[9] Y. Aoyama, G. Boutselis, A. Patel, and E. A. Theodorou, “Constrained differential dynamic programming revisited,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2021, pp. 9738–9744.

[10] V. Sindhwani, R. Roelofs, and M. Kalakrishnan, “Sequential operator splitting for constrained nonlinear optimal control,” in *American Control Conference (ACC)*, 2017, pp. 4864–4871.

[11] T. A. Howell, B. E. Jackson, and Z. Manchester, “Altro: A fast solver for constrained trajectory optimization,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 7674–7679.

[12] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization.” in *Proc. Robotics: Science and Systems (RSS)*, vol. 9, no. 1, 2013, pp. 1–10.

[13] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “Chomp: Gradient optimization techniques for efficient motion planning,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2009, pp. 489–494.

[14] E. G. Birgin, J. M. Martínez, and M. Raydan, “Spectral projected gradient methods: review and perspectives,” *Journal of Statistical Software*, vol. 60, pp. 1–21, 2014.

[15] E. Pignat, “Product of experts for robot learning from demonstration,” Ph.D. dissertation, EPFL, Lausanne, 2021.

[16] S. Calinon and D. Lee, “Learning control,” in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds. Springer, 2019, pp. 1261–1312.

[17] B. Settles, “Active learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[18] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, “Active incremental learning of robot movement primitives,” in *Conference on Robot Learning (CoRL)*, vol. 78, 2017, pp. 37–46.

[19] A. Conkey and T. Hermans, “Active learning of probabilistic movement primitives,” in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2019, pp. 1–8.

[20] T. S. Lembono and S. Calinon, “Probabilistic iterative lqr for short time horizon mpc,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 579–585.

[21] J. Anderson, J. C. Doyle, S. H. Low, and N. Matni, “System level synthesis,” *Annual Reviews in Control*, vol. 47, pp. 364–393, 2019.

[22] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, “Towards learning hierarchical skills for multi-phase manipulation tasks,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2015, pp. 1503–1510.

[23] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469 – 483, 2009.

[24] T. Osa, J. Pajarinen, and G. Neumann, *An Algorithmic Perspective on Imitation Learning.* Hanover, MA, USA: Now Publishers Inc., 2018.

[25] B. D. Ziebart, “Modeling purposeful adaptive behavior with the principle of maximum causal entropy,” Ph.D. dissertation, Machine Learning Department, Carnegie Mellon University, Dec 2010.

[26] B. Akgun and A. Thomaz, “Simultaneously learning actions and goals from demonstration,” *Autonomous Robots*, vol. 40, no. 2, pp. 211–227, Feb 2016.

[27] P. Englert, N. A. Vien, and M. Toussaint, “Inverse KKT: Learning cost functions of manipulation tasks from demonstrations,” *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1474–1488, 2017.

[28] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *Conference on Robot Learning (CoRL)*, 2016, pp. 49–58.

[29] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 4565–4573.

[30] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning for control,” in *Lazy learning.* Springer, 1997, pp. 75–113.

[31] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Learning attractor landscapes for learning motor primitives,” in *Advances in Neural Information Processing Systems (NIPS)*, 2002, pp. 1547–1554.

[32] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems (NIPS)*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. USA: Curran Associates, Inc., 2013, pp. 2616–2624.

[33] S. Calinon, F. Guenter, and A. G. Billard, “On learning, representing and generalizing a task in a humanoid robot,” *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 37, no. 2, pp. 286–298, 2007.

[34] T. Cederborg, L. Ming, A. Baranes, and P.-Y. Oudeyer, “Incremental local online Gaussian mixture regression for imitation learning of multiple tasks,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[35] S. M. Khansari-Zadeh and A. Billard, “Learning stable non-linear dynamical systems with Gaussian mixture models,” *IEEE Trans. on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[36] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[37] E. Pignat and S. Calinon, “Bayesian gaussian mixture model for robotic policy imitation,” *IEEE Robotics and Automation Letters*, Oct 2019.

[38] E. Pignat, T. Lembono, and S. Calinon, “Variational inference with mixture model approximation: Robotic applications,” 2019.

[39] O. Kroemer, S. Niekum, and G. D. Konidaris, “A review of robot learning for manipulation: Challenges, representations, and algorithms,” *J. Mach. Learn. Res.*, vol. 22, pp. 30:1–30:82, 2019.

[40] J. van den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, “Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2010, pp. 2074–2081.

[41] P. Englert, A. Paraschos, M. P. Deisenroth, and J. Peters, “Probabilistic model-based imitation learning,” *Adaptive Behavior*, vol. 21, no. 5, pp. 388–403, 2013.

[42] F. Torabi, G. Warnell, and P. Stone, “Recent advances in imitation learning from observation,” in *International Joint Conferences on Artificial Intelligence Organization*, 7 2019, pp. 6325–6331.

[43] ——, “Behavioral cloning from observation,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18.* International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 4950–4957.

[44] S. Calinon, *Robot Learning with Task-Parameterized Generative Models.* Springer International Publishing, 2018, vol. 3, pp. 111–126.

[45] H. Girgin and E. Ugur, “Associative skill memory models,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 6043–6048.

[46] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, “A survey on policy search algorithms for learning robot controllers in a handful of trials,” *IEEE Transactions on Robotics*, vol. 36, pp. 328–347, 2018.

[47] E. Pignat, H. Girgin, and S. Calinon, “Generative adversarial training of product of policies for robust and adaptive movement primitives,” in *Conference on Robot Learning.* PMLR, 2021, pp. 1456–1470.

[48] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[49] M. P. Deisenroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Found. Trends Robot*, vol. 2, pp. 1–142, Aug. 2013.

[50] D. A. Bristow, M. Tharayil, and A. G. Alleyne, “A survey of iterative learning control,” *IEEE control systems magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[51] O. Kroemer, R. Detry, J. Piater, and J. Peters, “Combining active learning and reactive control for robot grasping,” *Robotics and Autonomous systems*, vol. 58, no. 9, pp. 1105–1116, 2010.

[52] I. Abraham and T. D. Murphey, “Active learning of dynamics for data-driven control using koopman operators,” *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, 2019.

[53] S. Calinon, “Stochastic learning and control in multiple coordinate systems,” in *Intl Workshop on Human-Friendly Robotics*, Genova, Italy, 2016, pp. 1–5.

[54] K. Zhou, J. C. Doyle, K. Glover, *et al.*, *Robust and optimal control.* Prentice hall New Jersey, 1996, vol. 40.

[55] E. G. Birgin, J. Martinez, and M. Raydan, “Spectral projected gradient methods,” *Encyclopedia of Optimization*, vol. 2, 2009.

[56] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt, “On augmented lagrangian methods with general lower-level constraints,” *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1286–1309, 2008.

[57] E. G. Birgin and J. M. Martínez, *Practical augmented Lagrangian methods for constrained optimization.* SIAM, 2014.

[58] X. Jia, C. Kanzow, P. Mehlitz, and G. Wachsmuth, “An augmented lagrangian method for optimization problems with structured geometric constraints,” *Mathematical Programming*, pp. 1–51, 2022.

[59] M. Schmidt, E. Berg, M. Friedlander, and K. Murphy, “Optimizing costly functions with simple constraints: A limited-memory projected quasi-newton algorithm,” in *Artificial intelligence and statistics.* PMLR, 2009, pp. 456–463.

[60] H. H. Bauschke, “Projection algorithms and monotone operators,” Ph.D. dissertation, Simon Fraser University, 1996.

[61] H. H. Bauschke and P. L. Combettes, “Convex analysis and monotone operator theory in Hilbert spaces,” in *CMS Books in Mathematics*, 2011.

[62] I. Usmanova, M. Kamgarpour, A. Krause, and K. Levy, “Fast projection onto convex smooth constraints,” in *International Conference on Machine Learning.* PMLR, 2021, pp. 10 476–10 486.

[63] H. H. Bauschke and V. R. Koch, “Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces,” *Contemporary Mathematics*, vol. 636, pp. 1–40, 2015.

[64] J. P. Boyle and R. L. Dykstra, “A method for finding projections onto the intersection of convex sets in Hilbert spaces,” in *Advances in order restricted statistical inference.* Springer, 1986, pp. 28–47.

[65] G. Torrisi, S. Grammatico, R. S. Smith, and M. Morari, “A projected gradient and constraint linearization method for nonlinear model predictive control,” *SIAM Journal on Control and Optimization*, vol. 56, no. 3, pp. 1968–1999, 2018.

[66] M. Giftthaler and J. Buchli, “A projection approach to equality constrained iterative linear quadratic optimal control,” in *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017, pp. 61–66.

[67] M. I. Kamien and N. L. Schwartz, *Dynamic optimization: the calculus of variations and optimal control in economics and management.* Courier Corporation, 2012.

[68] F. D. Bianchi, H. De Battista, and R. J. Mantz, *Wind turbine control systems: principles, modelling and gain scheduling design.* Springer, 2007, vol. 19.

[69] M. Diehl, H. Bock, H. Diedam, and P.-B. Wieber, *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control.* Springer Berlin Heidelberg, 2006, pp. 65–93.

[70] V. Duchaine, S. Bouchard, and C. M. Gosselin, “Computationally efficient predictive robot control,” *IEEE/ASME Transactions on Mechatronics*, vol. 12, no. 5, pp. 570–578, 2007.

[71] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, “Biped walking pattern generation by using preview control of zero-moment point,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, vol. 2, 2003, pp. 1620–1626.

[72] S. Caron and A. Kheddar, “Multi-contact walking pattern generation based on model preview control of 3d com accelerations,” in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2016, pp. 550–557.

[73] B. Ponton, A. Herzog, A. Del Prete, S. Schaal, and L. Righetti, “On time optimization of centroidal momentum dynamics,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 1–7.

[74] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, “Gait and trajectory optimization for legged systems through phase-based end-effector parameterization,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1560–1567, 2018.

[75] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, “Differential dynamic programming for multi-phase rigid contact dynamics,” in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2018, pp. 1–9.

[76] D. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[77] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, “Whole-body model-predictive control applied to the hrp-2 humanoid,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3346–3351.

[78] N. Wintz and M. Bohner, “Linear quadratic tracker on time scales,” *International Journal of Dynamical Systems and Differential Equations*, vol. 3, pp. 423–447, 2011.

[79] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, “Trajectory optimization through contacts and automatic gait discovery for quadrupeds,” *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1502–1509, 2017.

[80] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, “Feedback mpc for torque-controlled legged robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4730–4737.

[81] M. P. Deisenroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Found. Trends Robot*, vol. 2, no. 1–2, p. 1–142, 2013.

[82] J. Siekmann, S. Valluri, J. Dao, F. Bermillo, H. Duan, A. Fern, and J. Hurst, “Learning memory-based control for human-scale bipedal locomotion,” in *Proc. Robotics: Science and Systems (RSS)*, 2020.

[83] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, “Learning deep neural network policies with continuous memory states,” *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 520–527, 2016.

[84] S. Dean, S. Tu, N. Matni, and B. Recht, “Safely learning to control the constrained linear quadratic regulator,” in *American Control Conference (ACC)*, 2019, pp. 5582–5588.

[85] S. Dean, N. Matni, B. Recht, and V. Ye, “Robust guarantees for perception-based control,” in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, vol. 120. PMLR, 2020, pp. 350–360.

[86] L. Jarin-Lipschitz, R. Li, T. Nguyen, V. Kumar, and N. Matni, “Robust, perception based control with quadrotors,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 7737–7743.

[87] D. Ho, “A system level approach to discrete-time nonlinear systems,” in *American Control Conference (ACC)*, 2020, pp. 1625–1630.

[88] J. Yu and D. Ho, “Achieving performance and safety in large scale systems with saturation using a nonlinear system level synthesis approach,” in *2020 American Control Conference (ACC)*, 2020, pp. 968–973.

[89] S. Calinon, “Gaussians on riemannian manifolds : Applications for robot learning and adaptive control,” *IEEE Robotics and Automation Magazine*, vol. 27, no. 2, pp. 33–45, 2020.

[90] T. A. Howell, S. L. Cleac’h, S. Singh, P. Florence, Z. Manchester, and V. Sindhwani, “Trajectory optimization with optimization-based dynamics,” *arXiv preprint arXiv:2109.04928*, 2021.

[91] E. Todorov and M. I. Jordan, “Optimal feedback control as a theory of motor coordination,” *Nature Neuroscience*, vol. 5, pp. 1226–1235, 2002.

[92] D. M. Wolpert, J. Diedrichsen, and J. R. Flanagan, “Principles of sensorimotor learning,” *Nature Reviews*, vol. 12, pp. 739–751, 2011.

[93] D. Sternad, S.-W. Park, H. Mueller, and N. Hogan, “Coordinate dependence of variability analysis,” *PLoS Comput. Biol.*, vol. 6, no. 4, pp. 1–16, 04 2010.

[94] G. Ganesh and E. Burdet, “Motor planning explains human behaviour in tasks with multiple solutions,” *Robotics and Autonomous Systems*, vol. 61, no. 4, pp. 362–368, 2013.

[95] I. D. Walker and S. I. Marcus, “Subtask performance by redundancy resolution for redundant robot manipulators,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 3, pp. 350–354, 1988.

[96] D. R. Baker and C. W. Wampler, “On the inverse kinematics of redundant manipulators,” *The International Journal of Robotics Research*, vol. 7, no. 2, pp. 3–21, 1988.

[97] B. Siciliano, “Kinematic control of redundant robot manipulators: A tutorial,” *Journal of intelligent and robotic systems*, vol. 3, no. 3, pp. 201–212, 1990.

[98] S. Chiaverini, “Kinematically redundant manipulators,” *Handbook of Robotics*, pp. 245–268, 2008.

[99] S. Calinon, I. Sardellitti, and D. G. Caldwell, “Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2010, pp. 249–254.

[100] F. Messerer, K. Baumgärtner, and M. Diehl, “Survey of sequential convex programming and generalized gauss-newton methods,” *ESAIM. Proceedings and Surveys*, vol. 71, p. 64, 2021.

[101] H. Hanafusa, T. Yoshikawa, and Y. Nakamura, “Analysis and control of articulated robot arms with redundancy,” *IFAC Proceedings Volumes*, vol. 14, no. 2, pp. 1927 – 1932, 1981, 8th IFAC World Congress on Control Science and Technology for the Progress of Society, Kyoto, Japan, 24-28 August 1981.

[102] B. Siciliano and J. . E. Slotine, “A general framework for managing multiple tasks in highly redundant robotic systems,” in *Fifth International Conference on Advanced Robotics ’Robots in Unstructured Environments*, June 1991, pp. 1211–1216 vol.2.

[103] P. Baerlocher and R. Boulic, “Task-priority formulations for the kinematic control of highly redundant articulated structures,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, vol. 1, Oct 1998, pp. 323–329 vol.1.

[104] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[105] S. Chiaverini, “Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators,” *IEEE Robotics and Automation Letters*, vol. 13, pp. 398 – 410, 07 1997.

[106] A. D. Prete, F. Romano, L. Natale, G. Metta, G. Sandini, and F. Nori, “Prioritized optimal control,” *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, pp. 2540–2545, 2014.

[107] F. R. Hogan and A. Rodriguez, “Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics,” *arXiv preprint arXiv:1611.08268*, 2016.

[108] J. P. De Moura, T. Stouraitis, and S. Vijayakumar, “Non-prehensile planar manipulation via trajectory optimization with complementarity constraints,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2022.

[109] D. Mayne, “A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems,” *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[110] N. Doshi, F. R. Hogan, and A. Rodriguez, “Hybrid differential dynamic programming for planar manipulation primitives,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 6759–6765.

[111] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2014, pp. 1168–1175.

[112] H. Girgin, J. Jankowski, and S. Calinon, "Reactive anticipatory robot skills with memory," in *International Symposium on Robotics Research (ISRR)*, 2022.

[113] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 2594–2601, 2020.

[114] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid mpc," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 247–253.

[115] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[116] P. Bonami, M. Kilinç, and J. Linderoth, "Algorithms and software for convex mixed integer nonlinear programs," in *Mixed integer nonlinear programming.* Springer, 2012, pp. 1–39.

[117] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 2536–2542.

[118] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[119] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 1–9.

[120] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[121] S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, "Using probabilistic movement primitives for striking movements," in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2016, pp. 502–508.

[122] G. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," *Autonomous Robots*, vol. 41, no. 3, pp. 593–612, 2017.

[123] M. Ewerton, G. Maeda, G. Kollegger, J. Wiemeyer, and J. Peters, “Incremental imitation learning of context-dependent motor skills,” in *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 2016, pp. 351–358.

[124] A. Paraschos, E. Rueckert, J. Peters, and G. Neumann, “Probabilistic movement primitives under unknown system dynamics,” *Advanced Robotics*, vol. 32, no. 6, pp. 297–310, 2018.

[125] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[126] E. Pignat and S. Calinon, “Bayesian Gaussian mixture model for robotic policy imitation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4452–4458, 2019.

[127] E. Pignat, J. Silvério, and S. Calinon, “Learning from demonstration using products of experts: Applications to manipulation and task prioritization,” *The International Journal of Robotics Research*, vol. 41, pp. 163 – 188, 2020.

[128] A. Sena, Y. Zhao, and M. Howard, “Teaching human teachers to teach robot learners.” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2018, pp. 5675–5681.

[129] A. Sena and M. Howard, “Quantifying teaching behavior in robot learning from demonstration,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 54–72, 2020.

[130] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[131] M. Salganicoff, L. Ungar, and R. Bajcsy, “Active learning for vision-based robot grasping,” *Machine Learning*, vol. 23, no. 2-3, pp. 251–278, 1996.

[132] S. Ivaldi, N. Lyubova, A. Droniou, D. Gerardeaux-Viret, D. Filliat, V. Padois, O. Sigaud, P. Oudeyer, *et al.*, “Learning to recognize objects through curiosity-driven manipulation with the icub humanoid robot,” in *Proc. IEEE Intl Conf. on Development and Learning and Epigenetic Robotics (ICDL)*, 2013, pp. 1–8.

[133] J. Schmidhuber, “Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts,” *Connection Science*, vol. 18, no. 2, pp. 173–187, 2006.

[134] A. Shon, D. Verma, and R. Rao, “Active imitation learning,” in *Proc. AAAI Conference on Artificial Intelligence*, 2007, pp. 1–7.

[135] D. Silver, J. Bagnell, and A. Stentz, “Active learning from demonstration for robust autonomous navigation,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2012, pp. 200–207.

[136] C. Chao, M. Cakmak, and A. Thomaz, "Transparent active learning for robots," in *Proc. ACM/IEEE Intl Conf. on Human-Robot Interaction (HRI)*, 2010, pp. 317–324.

[137] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, pp. 1–25, 2009.

[138] F. Nielsen, "Closed-form information-theoretic divergences for statistical mixtures," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE, 2012, pp. 1723–1726.

[139] C. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[140] A. Kolchinsky and B. Tracey, "Estimating mixture entropy with pairwise distances," *Entropy*, vol. 19, no. 7, p. 361, 2017.

[141] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyperparameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[142] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13, 2013, p. 20.

[143] S. A. Bowyer, B. L. Davies, and F. Rodriguez y Baena, "Active constraints/virtual fixtures: A survey," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 138–157, 2014.

[144] S. Zimmermann, M. Busenhart, S. Huber, R. Poranne, and S. Coros, "Differentiable collision avoidance using collision primitives," in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 2022, pp. 8086–8093.

[145] K. M. Lynch and F. C. Park, *Modern robotics.* Cambridge University Press, 2017.

[146] N. Parikh, S. Boyd, *et al.*, "Proximal algorithms," *Foundations and trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[147] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2021.

[148] M. Koptev, N. Figueroa, and A. Billard, "Neural joint space implicit signed distance functions for reactive robot manipulator control," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 480–487, 2022.

[149] N. Jaquier, R. Haschke, and S. Calinon, "Tensor-variate mixture of experts for proportional myographic control of a robotic hand," *Robotics and Autonomous Systems*, vol. 142, p. 103812, 2021.

[150] J. Peters and S. Schaal, "Learning to control in operational space," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2008.

[151] B. Boots, G. J. Gordon, and S. M. Siddiqi, "A constraint generation approach to learning stable linear dynamical systems," in *Advances in Neural Information Processing Systems (NIPS).* Curran Associates, Inc., 2008, pp. 1329–1336.

[152] A. Venkatraman, R. Capobianco, L. Pinto, M. Hebert, D. Nardi, and J. A. Bagnell, "Improved learning of dynamics models for control," in *International Symposium on Experimental Robotics.* Springer, 2016, pp. 703–713.

[153] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proc. Intl Conf. on Machine Learning (ICML)*, 2011, pp. 465–472.

[154] M. Kopicki, S. Zurek, R. Stolkin, T. Moerwald, and J. L. Wyatt, "Learning modular and transferable forward models of the motions of push manipulated objects," *Autonomous Robots*, vol. 41, no. 5, pp. 1061–1082, 2017.

[155] E. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky, "Nonparametric Bayesian learning of switching linear dynamical systems," in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 457–464.

[156] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proc. National Academy of Science*, vol. 17, no. 5, p. 315, 1931.

[157] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven approximations of dynamical systems operators for control," in *The Koopman Operator in Systems and Control.* Springer, 2020, pp. 197–234.

[158] N. Takeishi, Y. Kawahara, Y. Tabei, and T. Yairi, "Bayesian dynamic mode decomposition," in *International Joint Conference on Artificial Intelligence*, 2017.

HAKAN GIRGIN

☎ +41 76 506 9890 | ✉ hakan.girgin@epfl.ch | hgirgin.github.io
Rue de l'ancien-stand 8, Montreux, Switzerland

PROFILE

I am currently Ph.D. candidate in robotics at Ecole Polytechnique Fédérale de Lausanne (EPFL) and Idiap Research Institute supervised by Dr. Sylvain Calinon. My research focuses on optimization and active iterative refinement of feedback and feedforward control policies for acquiring robust and anticipatory robot skills from demonstration.

EDUCATION

Ph.D. | ***Robotics*** 2018 – 2023
Ecole Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland

Bachelor of Science | ***Mechanical Engineering*** 2012 – 2017
Bogazici University, Entrance rank: 0.13%, GPA: **3.90/4.0** Istanbul, Turkey

Exchange semester | ***Mechanical Engineering*** 2014 – 2015
Ecole Centrale Paris Paris, France

Francophone Highschool 2007 – 2012
Lycée de Galatasaray, Entrance rank: 0.06% GPA: **85.84/100** Istanbul, Turkey

WORK EXPERIENCE

Ph.D. Research Assistant 2018 – 2023
Idiap Research Institute, Robot Learning and Interaction Group Martigny, Switzerland

- Ph.D. research supervised by Dr. Sylvain Calinon
- Worked on EU-Horizon2020 project CoLLaboratE

Teaching Assistant 2020-2022
Robotics Course, AI-Master Programme, Unidistance Martigny, Switzerland

- Preparation of exercises in Jupyter notebooks with ROS and in html formats

Bachelor Research Assistant 2017 – 2018
Bogazici University Cognitive Robotics and Learning Systems Lab (CoLoRs) Istanbul, Turkey

- Assistance to Dr. Emre Ugur in forming CoLoRs lab
- Worked on EU-Horizon 2020 project, IMAGINE

Product Definition Engineering Intern 2015 – 2016
General Electric Aviation Istanbul, Turkey

System Engineering Intern 2016
ALTINAY Aerospace & Advanced Technologies Istanbul, Turkey

SKILLS

Languages: Turkish (Native), English (Proficient), French (Proficient), Japanese (B1), Italian (A1), German (A1)

Programming: Python, Jupyter, MATLAB, Tensorflow, PyTorch, Pyscript

Software: ROS, PyBullet, KDL

PUBLICATIONS

Demonstration-guided Optimal Control for Long-term Non-prehensile Planar Manipulation
T. Xue, H. Girgin, T. Lembono, S. Calinon, In Proc. IEEE Intl Conf. on Robotics and Automation ICRA 2023

Reactive Anticipatory Robot Skills with Memory
H. Girgin, J. Jankowski, S. Calinon, International Symposium on Robotics Research ISRR 2022

Optimization of robot configurations for motion planning in industrial riveting
H. Girgin, T. Lembono, R. Cirligeanu, S. Calinon, In Proc. IEEE Intl Conf. on Advanced Robotics ICAR 2021

Active Learning of Bayesian Probabilistic Movement Primitives
T. Kulak, H. Girgin, J.-M. Odobez, S. Calinon, IEEE Robotics and Automation Letters RAL 2021

Probabilistic Adaptive Control for Robust Behavior Imitation
J. Jankowski, H. Girgin, S. Calinon, IEEE Robotics and Automation Letters RAL 2021

Active Improvement of Control Policies with Bayesian Gaussian Mixture Model
H. Girgin, E. Pignat, N. Jaquier, S. Calinon, IEEE Intl. Conf. on Intelligent Robots and Systems IROS 2020

Generative Adversarial Training of Product of Policies for Robust and Adaptive Movement Primitives
E. Pignat, H. Girgin, S. Calinon, In Proc. Conference on Robot Learning CoRL 2020

Compliant Parametric Dynamic Movement Primitives
E. Ugur, H. Girgin, Robotica, 38(3), pp. 457-474 Robotica 2020

Associative Skill Memory Models
H. Girgin, E. Ugur, IEEE Intl. Conf. on Intelligent Robots and Systems IROS 2018