UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Bimanual robot skills: MP encoding, dimensionality reduction and reinforcement learning

Adrià Colomé Figueras

Universitat Politècnica de Catalunya

Doctoral Programme

Automatic Control, Robotics and Computer Vision

Ph.D. Thesis

Bimanual robot skills: MP encoding, dimensionality reduction and reinforcement learning

Adrià Colomé Figueras

Advisor:
Carme Torras

Barcelona, May 2017

Bimanual robot skills: MP encoding, dimensionality reduction and reinforcement learning

A thesis submitted to the Universitat Politècnica de Catalunya
to obtain the degree of Doctor of Philosophy

Doctoral programme:
Automatic Control, Robotics and Computer Vision

This thesis was completed at:
Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Thesis advisor:
Carme Torras

To Ivet

BIMANUAL ROBOT SKILLS:
MP ENCODING, DIMENSIONALITY REDUCTION
AND REINFORCEMENT LEARNING

Adrià Colomé Figueras

Abstract

In our culture, robots have been in novels and cinema for a long time, but it has been specially in the last two decades when the improvements in hardware - better computational power and components - and advances in Artificial Intelligence (AI), have allowed robots to start sharing spaces with humans. Such situations require, aside from ethical considerations, robots to be able to move with both compliance and precision, and learn at different levels, such as perception, planning, and motion, being the latter the focus of this work.

The first issue addressed in this thesis is inverse kinematics for redundant robot manipulators, i.e: positioning the robot joints so as to reach a certain end-effector pose. We opt for iterative solutions based on the inversion of the kinematic Jacobian of a robot, and propose to filter and limit the gains in the spectral domain, while also unifying such approach with a continuous, multipriority scheme. Such inverse kinematics method is then used to derive manipulability in the whole workspace of an antropomorphic arm, and the coordination of two arms is subsequently optimized by finding their best relative positioning.

Having solved the kinematic issues, a robot learning within a human environment needs to move compliantly, with limited amount of force, in order not to harm any humans or cause any damage, while being as precise as possible. Therefore, we developed two dynamic models for the same redundant arm we had analysed kinematically: The first based on local models with Gaussian projections, and the second characterizing the most problematic term of the dynamics, namely friction. Such models allowed us to implement feed-forward controllers, where we can actively change the weights in the compliance-precision tradeoff. Moreover, we used such models to predict external forces acting on the robot, without the use of force sensors.

Afterwards, we noticed that bimanual robots must coordinate their components (or limbs) and be able to adapt to new situations with ease. Over the last decade, a number of successful applications for learning robot motion tasks have been published. However, due to the complexity of a complete system including all the required elements, most of these applications involve only simple robots with a large number of high-end technology sensors, or consist of very simple and controlled tasks. Using our previous framework for kinematics and control, we relied on two types of movement primitives to encapsulate robot motion. Such movement primitives are very suitable for using reinforcement learning. In particular, we used direct policy search, which uses the motion parametrization as the policy itself.

In order to improve the learning speed in real robot applications, we generalized a policy search algorithm to give some importance to samples yielding a bad result, and we paid special attention to the dimensionality of the motion parametrization. We reduced such dimensionality with linear methods, using the rewards obtained through motion repetition and execution. We tested such framework in a bimanual task performed by two antropomorphic arms, such as the folding of garments, showing how a reduced dimensionality can provide qualitative information about robot couplings and help to speed up the learning of tasks when robot motion executions are costly.

Keywords: Robotics, Artificial Intelligence, Reinforcement Learning, Redundant Robots,Robot Kinematics, Robot Dynamics, Robot Motion Characterization, Dimensionality Reduction.

Resum

A la nostra cultura, els robots han estat presents en novel·les i cinema des de fa dècades, però ha sigut especialment en les últimes dues quan les millores en hardware (millors capacitats de còmput) i els avenços en intel·ligència artificial han permès que els robots comencin a compartir espais amb els humans. Aquestes situacions requereixen, a banda de consideracions ètiques, que els robots siguin capaços de moure's tant amb suavitat com amb precisió, i d'aprendre a diferents nivells, com són la percepció, planificació i moviment, essent l'últim el centre d'atenció d'aquest treball.

El primer problema tractat en aquesta tesi és la cinemàtica inversa, i.e.: posicionar les articulacions del robot de manera que l'efector final estigui en una certa posició i orientació. Hem estudiat el camp de les solucions iteratives, basades en la inversió del Jacobià cinemàtic d'un robot, i proposem un filtre que limita els guanys en el seu domini espectral, mentre també unifiquem tal mètode dins un esquema multi-prioritat i continu. Aquest mètode per a la cinemàtica inversa és usat a l'hora d'encapsular tota la informació sobre l'espai de treball d'un braç antropomòrfic, i les capacitats de coordinació entre dos braços són optimitzades, tot trobant la seva millor posició relativa en l'espai.

Havent resolt les dificultats cinemàtiques, un robot que aprèn en un entorn humà necessita moure's amb suavitat exercint unes forces limitades per tal de no causar danys, mentre es mou amb la màxima precisió possible. Per tant, hem desenvolupat dos models dinàmics per al mateix braç robòtic redundant que havíem analitzat des del punt de vista cinemàtic: El primer basat en models locals amb projeccions de Gaussianes i el segon, caracteritzant el terme més problemàtic i difícil de representar de la dinàmica, la fricció. Aquests models ens van permetre utilitzar controladors coneguts com *feed-forward*, on podem canviar activament els guanys buscant l'equilibri precisió-suavitat que més convingui. A més, hem usat aquests models per a inferir les forces externes actuant en el robot, sense la necessitat de sensors de força.

Més endavant, ens hem adonat que els robots bimanuals han de coordinar els seus components (braços) i ser capaços d'adaptar-se a noves situacions amb facilitat. Al llarg de l'última dècada, diverses aplicacions per aprendre tasques motores robòtiques amb èxit han estat publicades. No obstant, degut a la complexitat d'un sistema complet que inclogui tots els elements necessaris, la majoria d'aquestes aplicacions consisteixen en robots més aviat simples amb costosos sensors d'última generació, o a resoldre tasques senzilles en un entorn molt controlat. Utilitzant el nostre treball en cinemàtica i control, ens hem basat en dos tipus de primitives de moviment per caracteritzar la motricitat robòtica. Aquestes primitives de moviment són molt adequades per usar aprenentatge per reforç. En particular, hem usat la búsqueda directa de la política, un camp de l'aprenentatge per reforç que usa la parametrització del moviment com la pròpia política.

Per tal de millorar la velocitat d'aprenentatge en aplicacions amb robots reals, hem generalitzat un algoritme de búsqueda directa de política per a donar importància a les mostres amb mal resultat, i hem posat especial atenció a la reducció de dimensionalitat en la parametrització dels moviments. Hem reduït la dimensionalitat amb mètodes lineals, utilitzant les recompenses obtingudes a l'executar els moviments. Aquests mètodes han estat provats en tasques bimanuals com plegar roba, usant dos braços antropomòrfics. Els resultats mostren com la reducció

de dimensionalitat pot aportar informació qualitativa d'una tasca, i al mateix temps ajuda a aprendre-la més ràpid quan les execucions amb robots reals són costoses.

Acknowledgements

I would like to thank my advisor Prof. Carme Torras, who taught me how to do research and guided me through this years. Her help has been crucial for my work.

Besides, I would also like to thank Dr. Guillem Alenyà and Dr. Sergi Foix for their help and support in an uncountable number of occasions, specially in the Perception and Manipulation Lab. I want to also thank all my colleagues at D19 and IRI for all the good times spent toghether (and more to come).

My sincere thanks also to Prof. Jan Peters and all the people at the IAS Lab in Darmstadt, for welcoming me to work with them during my stay.

I would also like to thank my parents and Raquel for their continuous support through these years. And my daughter Ivet, whose smile brought a new meaning of happiness to me.

I wish I could also thank all those relatives that left me through the years. The void they left will be never filled again. People go, but memories remain.

This work has been supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant **AP2010-1989**.

This thesis has also been partially supported by the research projects:

FP7-ICT-2009-6-269959 IntellAct: Intelligent observation and execution of Actions and manipulations (European Project).

201350E102 MANIPlus: Manipulación robotizada de objetos deformables (CSIC Project).

2009 SGR 155 : SGR ROBÒTICA: Grup de recerca consolidat - Grup de Robòtica (Catalan government).

TIN2014-58178-R RobInstruct: Instructing robots using natural communication skills (Spanish national project).

PCIN-2015-147 I-DRESS: Assistive interactive robotic system for support in dressing (European Project).

The latest version of this thesis can be obtained at `https://www.iri.upc.edu/staff/acolome`.

Contents

Abstract **v**

Resum **vii**

Acknowledgements **ix**

1 Introduction **1**
1.1 Objectives . . . 3
1.2 Contributions . . . 4
1.3 Outline . . . 5

I Compliant Redundant Robot Control **11**

2 State of the Art **13**
2.1 Inverse kinematics of redundant serial robots . . . 14
2.2 Bimanual manipulation . . . 18
2.3 Compliant control and wrench estimation . . . 19
2.3.1 Learning robot inverse dynamics . . . 21
2.3.2 Fitting an analytical model . . . 23
2.4 Summary . . . 25

3 Inverse Kinematics and Relative Arm Positioning **27**
3.1 Inverse kinematics of redundant robots . . . 27
3.1.1 CLIK algorithm issues . . . 27
3.1.2 First enhancement: Singular Value Filtering (SVF) . . . 31
3.1.3 Multiple tasks . . . 35
3.1.4 Joint limit avoidance . . . 36
3.1.5 Second enhancement: pseudoinverse smoothing . . . 39
3.1.6 Experimentation . . . 41
3.1.7 Discussion . . . 45
3.2 Bimanual arm positioning . . . 47
3.2.1 Workspace representation . . . 48
3.2.2 Bimanual workspace . . . 50
3.2.3 Proposed quality function . . . 50
3.2.4 Experimentation . . . 53
3.3 Summary . . . 54

4 Robot Compliant Control **57**
4.1 External force estimation . . . 58
4.1.1 External wrench estimation as a disturbance observer . . . 59
4.1.2 Experimentation . . . 63
4.1.3 Discussion . . . 66
4.2 Building a friction model . . . 67

4.2.1 Introduction . . . 68
4.2.2 Advanced model for the WAM robot . . . 68
4.3 Applications . . . 72
4.3.1 Scarf-placing experiment . . . 72
4.3.2 Compliant object tracking . . . 75
4.4 Summary . . . 76

II Reinforcement Learning with Movement Primitives 79

5 Preliminaries 81
5.1 Policy Search (PS) . . . 81
5.1.1 Robot control as a reinforcement learning problem . . . 82
5.2 Movement Primitives (MP) . . . 86
5.2.1 Dynamic movement primitives . . . 86
5.2.2 Probabilistic movement primitives . . . 89
5.3 Summary . . . 92

6 Dual REPS: A Generalization of Relative Entropy Policy Search Exploiting Bad Experiences 93
6.1 The Dual REPS Algorithm . . . 94
6.1.1 Clustering . . . 95
6.1.2 DREPS derivation . . . 97
6.1.3 Experiments . . . 103
6.2 Summary . . . 111

7 Reward-oriented Dimensionality Reduction with Movement Primitives 113
7.1 Dimensionaliy Reduction for ProMPs . . . 115
7.1.1 Representing Dimensionality Reduction for ProMP (DR-ProMP) . . . 116
7.1.2 DR-ProMP for robot control . . . 118
7.1.3 Fitting DR-ProMP parameters with expectation maximization . . . 119
7.1.4 Experiments . . . 124
7.1.5 Conclusions . . . 128
7.2 Dimensionality Reduction for DMPs . . . 128
7.2.1 DMP coordination . . . 129
7.2.2 EM approach to find the latent space projection . . . 137
7.2.3 Experimentation . . . 139
7.2.4 Conclusions . . . 149
7.3 Summary . . . 150

8 Conclusions 151
8.1 Summary . . . 151
8.2 Future work . . . 153
8.3 Epilogue . . . 155

A List of Publications 157

B External Resources **159**
B.1 Closed-loop inverse kinematics for redundant robots 159
B.2 Friction model applications 159
B.3 Realtime tracking and grasping of a moving object from range video 159
B.4 Human-guided compliant control 159
B.5 DREPS 159
B.6 DR for DMPs 159

C Acronyms **161**

D Glossary **163**

Bibliography **165**

Figures

1.1 Two Barrett's WAM robots holding a piece of cloth . . . 5
1.2 Global thesis scheme . . . 6
1.3 Thesis overview . . . 7

2.1 Kinesthetic teaching of a robot . . . 19
2.2 Barrett WAM robot holding cloth garments . . . 21
2.3 Example data used as input . . . 23

3.1 Singular values inversion in Jacobian damping algorithms . . . 30
3.2 Condition number comparison for different algorithms . . . 33
3.3 CLIK algorithms behavior for a 4R planar manipulator trajectory (1/2) . . . 34
3.4 CLIK algorithms behavior for a 4R planar manipulator trajectory (2/2) . . . 42
3.5 Solution finding success ratio of different CLIK algorithms . . . 46
3.6 End effector bounding cones . . . 49
3.7 IK solution density map . . . 49
3.8 Orientation factor for two cones . . . 52
3.9 Experimental Settings results . . . 53
3.10 Experimental Settings results . . . 54

4.1 Force estimation scheme . . . 58
4.2 Cloth grasp detection experiment results . . . 65
4.3 Estimated torques for a fixed load ($1kg$) throughout a trajectory . . . 66
4.4 Estimated torques for a fixed load ($0.5kg$) throughout a trajectory . . . 67
4.5 Feed forward control scheme with force estimation . . . 68
4.6 Friction models comparison . . . 71
4.7 Scarf detection examples . . . 73
4.8 Learning curve for the scarf experiment . . . 73
4.9 Scarf around a mannequin's neck . . . 74
4.10 Visual tracking scheme . . . 75
4.11 WAM tracking an object . . . 76

5.1 Gaussian basis function example . . . 87
5.2 Modified Gaussian basis functions . . . 88
5.3 Phase variable . . . 88
5.4 ProMP example obtained from a set of trajectories . . . 90

6.1 K-means clustering example . . . 96
6.2 Reward function for clustering example . . . 98
6.3 K-means example clustering result . . . 98
6.4 Proposed clustering algorithm example result . . . 99
6.5 Learning curve of REPS in a multiple solution problem . . . 104
6.6 REPS averaging between optimal solutions . . . 104
6.7 Learning curves of DREPS vs REPS . . . 105
6.8 Real robot experimental setup for the bottle task . . . 106

6.9 Time alignment of several trajectories . . . 107
6.10 Learning curves of DREPS vs REPS for the bottle task . . . 108
6.11 Example trajectories obtained for the bottle task . . . 109
6.12 Learning curves of DREPS vs REPS with new reward function . . . 109
6.13 Solution comparison for bottle task . . . 110
6.14 DREPS vs REPS in unimodal problem . . . 111

7.1 NAO robot in a human environment . . . 116
7.2 KL divergence comparison between EM and PCA . . . 124
7.3 NAO correlation values . . . 125
7.4 NAO walking trajectory distribution . . . 126
7.5 Different learning approaches on a planar manipulator task . . . 128
7.6 10-DoF planar arm experiment with DR-DMPs . . . 142
7.7 7-DoF WAM robot experiment with DR-DMPs . . . 143
7.8 Two robot arms . . . 146
7.9 14-DoF dual-arm real-robot experiment with DR-DMPs . . . 147

Tables

3.1 Methods abbreviations 28
3.2 Reprojection error. 33
3.3 WAM Denavit-Hartenberg parameters 43
3.4 Joint limits unconcerned IK success for CLIK algorithms 44
3.5 Joint limits concerned IK success for CLIK algorithms 44

4.1 Friction Validation Results 70

7.1 Methods description 137
7.2 Methods initialization and usage 137
7.3 Results for the 10-DoF planar arm experiment 141
7.4 Results for the real data simulated 7-DoF WAM experiment 144
7.5 Results for the dual-arm real-robot experiment of folding clothes 149

C.1 Acronym list 162

D.1 Relevant variables (Part I) 163
D.2 Relevant variables (Part II) 164

1

Introduction

Back to the early-mid XXth century, the neurophysiologist Nikolai A. Bernstein studied human movement during manual labor. His aim was to track human movement in order to help to improve productivity. Bernstein was one of the pioneers in the fields of human motor control and learning, claiming that "we humans activate in a coordinated manner those muscles that we cannot control individually" [1]. The large amount of muscles in a human body is considered to be around 640 [2], which is a number of degrees of freedom a human brain cannot possibly control independently. Such muscles are responsible for the diverse motions of the human skeleton by using ligaments and tendons. Therefore, humans learn synergy patterns associated to each task they want to perform. These patterns are then stored in the human brain and executed when needed, and reevaluated after every execution in order to improve further. Such is the example of a tennis player learning his swing. He will be first imitating someone else's swing to later improve through experience: every time the swing motion is executed, the player is likely to explore and innovate, and the outcome of each execution is evaluated. If such outcome was considered positive, the muscle synergy stored in the human's brain is modified. Otherwise, the applied changes are discarded.

In parallel, it was back in the 1920's when writer Karel Čapek defined the term robot, coming from the Czech term *robota* - forced labor- in his play *Rossum's Universal Robots*. Čapek imagined a world where humans created artificial living creatures to force them to work at factories, but robots later rebelled against humanity causing their extinction. Since Čapek's play, robots became progressively more popular in science fiction magazines published in the XXth century. In those stories, robots were often portrayed as part of our lives, as a working force or household assistants. Moreover, the popularization of mass production, together with the automatization of industry resulted in an increase in production capability. It is then, when in 1960's, the scientific community started to focus on providing robots with more intelligence and autonomy. In an approach to emulate the human way of learning, the field of Reinforcement Learning (RL)

became popular, specially in the last two decades of the XXth century, after Richard S. Sutton and Andrew G. Barto [3] defined RL as a new science field. Since then, RL has been developing in several subfields. Nowadays, RL is a wide field of artificial intelligence used in a large variety of applications, from big data analysis to planning and robot motion.

Today, the industry is creating better and more accessible robots every year. These robots require a good kinematics and dynamics knowledge [4], to be able to position themselves where they need to, and to move in a safe manner in the case of a a human environment. However, solving the inverse kinematics chain in order to find the proper joint positions for placing a robot's end-effector in a certain pose is a complex problem for redundant manipulators such as anthropomorphic arms. Iterative algorithms are a popular choice for its generality, but at the risk of suffering from numerical stability when close to a robot singular position. Moreover, joint limits strongly restrict the feasible solutions and need to be taken into account. Once the kinematic chain is solved, the control problem needs to be taken into consideration, specially in the case of unmodelled scenarios, including interaction with humans and/or deformable objects. It is recommended that robots have a so-called compliant controller, i.e.: a control scheme that is able to precisely track a desired command while exerting minimal torques, in order not to harm any object/human in the loop. Such controllers require feedback from the robotic sensors, but in some cases, only the torque commands and joint position encoders are available.

In addition, it is also natural to expect those robots to be capable of learning and reasoning. In particular, learning motor skills in a similar way humans do. Learning robotic skills is a difficult problem which can be addressed in several ways. The most common approach is Learning from Demonstration (LfD), in which the robot is shown an initial way of solving a task, and then tries to reproduce, improve and/or adapt it to variable conditions. The learning of tasks is usually performed in the kinematic domain by learning trajectories [5–7], but it can also be done in the force domain [8,9]. A training data set is often used in order to fit a relation between an input (experiment conditions) and an output (a good behavior of the robot). This fitting, which can use different regression models such as Gaussian Mixture Models (GMM) [10], is then adapted to the environmental conditions in order to modify the robot's behavior [11]. However, reproducing the demonstrated behavior and adapting it to new situations does not always solve a task optimally, thus Reinforcement Learning (RL) is also being used, where the solution learned from a demonstration, improves through exploratory trial-and-error. RL is capable of finding better solutions than the one demonstrated to the robot.

The latest approaches in motor skills are based on motion parametrization combined with direct Policy Search (PS), a particular subfield of RL, where policies - the agents deciding which actions to take or which motions to perform - are characterized as a set of parameters, that are optimized after locally exploring variants of similar motions and evaluating them with a

reward function [12]. Such PS methods suffer, though, from several complications, like a limited amount of samples in real-robot executions, that need to be minimized for cost reasons. The rise of complexity of those robots requires to deal with more Degrees of Freedom (DoF) and, subsequently, more parameters in the motion representation. This large number of parameters leads to the known curse of dimensionality, defined by Bellman in 1957 [13], i.e.: meaningful information is *lost* within a large-dimensional learning space. Such dimensionality is treated by learning algorithms by using *greedy* policy updates, meaning that only the known data is considered, and exploration into the unknown is limited [12], resulting in an often suboptimal solution of the learning process. To overcome this limitation, it is therefore necessary to apply a similar approach to the one Bernstein defined: to create actuator synergies in order to be able to easily control, encode, and execute robot motion tasks. This is the main focus of this work.

In this thesis, we will discuss and tackle all the problems associated to the learning of motor tasks by complex robots - e.g., bimanual robots- in an often unstructured or unmodelable environment, as it can be the task of folding clothes. Due to limited amount of data samples in robotic learning scenarios, such tasks require dimensionality reduction techniques so that such complex problems can be solved. The scope of this work is limited to serial robots, learning tasks at the motion level. Grasping and/or planning are not focused on, as those can be built independently.

1.1 Objectives

The ultimate objective of this thesis is to build a **complete framework for a model-free, compliant, coordinated robot motion learning** scenario. Such a framework requires the fulfillment of a series of secondary objectives, namely:

- Working on **robot motion learning** to apply the latest advances in Reinforcement Learning (RL) and its subfield of policy search to learn robotic motion, also requiring a proper motion characterization.

- Build a **complete framework** by developing all levels (kinematics, dynamics, motion characterization and learning) under a unified perspective. Such levels can relate and benefit from each other, as kinematics is required for motion characterization and dynamics. Dynamics also need to be used for learning, e.g., taken into account in the reward function.

- Using **model-free** alternatives from the learning perspective, since human environments and/or cloth objects are very costly to model. While the robot's dynamics can be modeled

up to a certain error, the environment and experimental setup may not be possible to model and, therefore, model-based reinforcement learning is out of the scope of this thesis.

- The controller should be as **compliant** as possible, since we assume fragile and/or deformable objects, as well as humans, might be in the scenario and, therefore, robot stiffness and other safety issues need to be taken into account.

- the framework must permit **coordinating** the robot's DoF and/or the arms, since their motion should not be controlled independently, but coupled.

1.2 Contributions

This thesis has contributed to:

1. Review, evaluate and improve some of the most popular algorithms available for computing the **inverse kinematics** of redundant manipulators. In particular, we reviewed the most used Closed-Loop Inverse Kinematics (CLIK) algorithms in [14] and [15]. CLIK algorithms are based on inverting the Jacobian matrix of a robot manipulator. Such matrix inversion can be ill-conditioned, resulting in numerical instability, or generate oscillations around a solution. To prevent these situations, we propose a new way of filtering the Jacobian matrix in the spectral domain for its inversion, while limiting the gains and also using a matrix inversion which is continuous wrt. the changes of the rank of the matrix to be inverted. This results in a more robust solution, that we tested in a real robot (see video B.1 in Appendix B). For a setup with two Whole Arm Manipulator [16] (WAM) robots (see Fig. 1.1), we used these inverse kinematics mappings and an encapsulation of all the possible orientation solutions for each Cartesian end-effector position, to evaluate and optimize the combined manipulability for the relative positioning of the two arms [17], allowing to optimize such relative positioning under user-specified restrictions.

2. We modeled the **dynamics** of a WAM robot with locally fitted models, and used such models to detect external contact forces with the robot when moving, by using joint position encoders and torque commands only [18] [19]. Such forces are then used for contact detection or in RL algorithms to obtain more information about experiments in model-free setups, as well as to allow for compliant manipulation [20] (see video B.3 in Appendix B). We then went on building a global friction model for the WAM robot, allowing for a more precise dynamics model that could be used all throughout the robot's workspace [21] (see video B.2 in Appendix B).

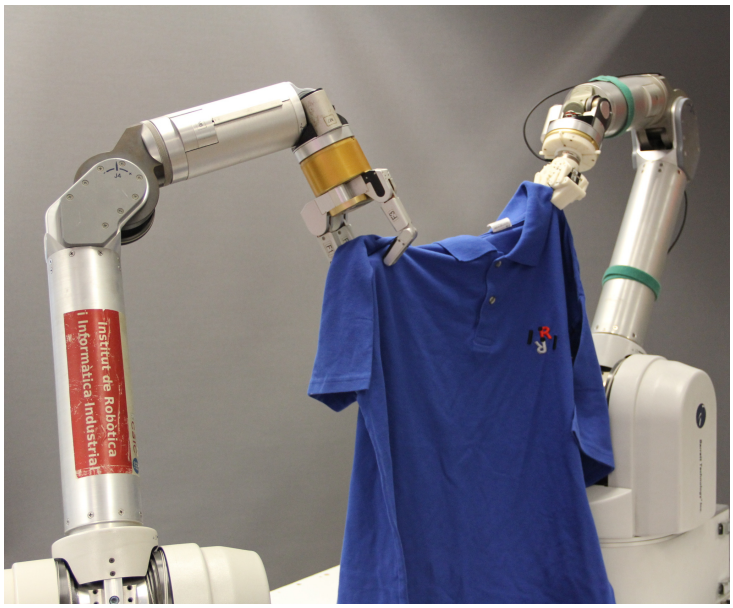


Figure 1.1: Two Barrett's WAM robots holding a piece of cloth.

3. We implemented and used PS algorithms, such as Relative Entropy Policy Search (REPS), which we generalized by allowing it to reuse bad-performing experiences as a repulsor in terms of probability distributions by forcing a difference between the bad samples' clustered distribution and the optimized solution. We named **Dual REPS** (DREPS) such generalization. Some results and real-robot videos are shown in the experimentation section and in video B.5 in Appendix B. We also applied DREPS to human environments, using hybrid methods for gathering data, by visual imitation or randomly-generated samples in [22, 23]. Video B.4 in Appendix B show this hybrid approach.

4. We used the preceding items to perform reward-oriented linear **dimensionality reduction** in a RL framework to learn robot motion faster. We used Movement Primitives (MP), such as Probabilistic Movement Primitives (ProMPs) or Dynamic Movement Primitives (DMPs), to characterize motion. Then, with linear dimensionality reduction techniques on the robot's DoF, we obtained reduced motion characterizations for both DMPs [24–26] (see video B.6 in Appendix B) and ProMPs [27].

The whole process of learning robot motion skills can seen as a block diagram in Fig. 1.2.

1.3 Outline

This thesis is divided in two parts, which are related as seen in Fig. 1.3.

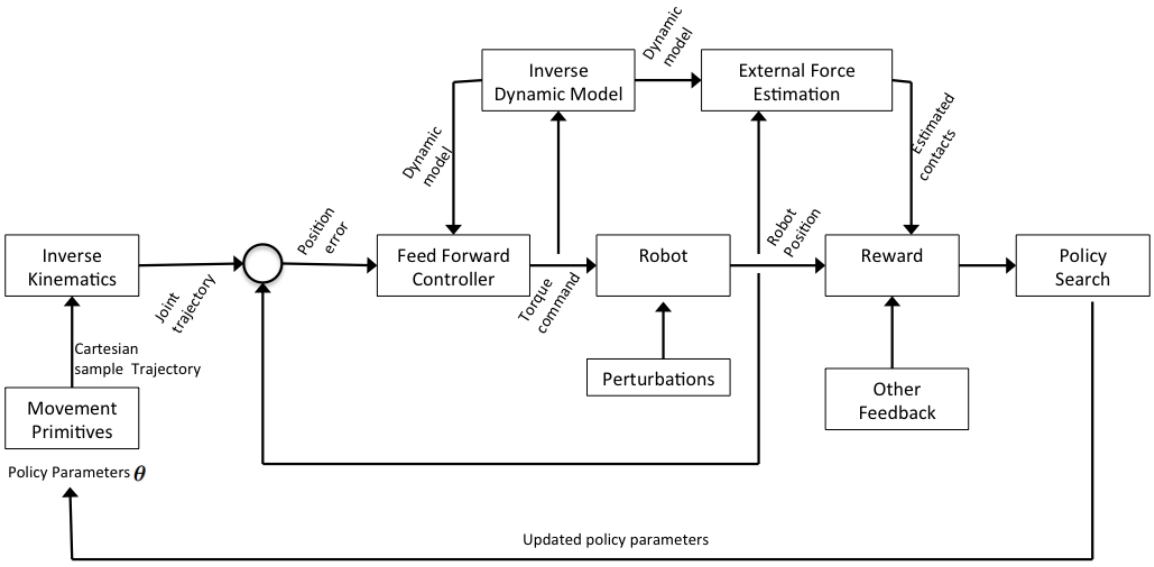


Figure 1.2: Thesis block diagram. A motion is encoded by a movement primitive, storing the policy parameters $\boldsymbol{\theta}$. A sample trajectory is obtained from such policy and, if necessary, translated to a joint trajectory through inverse kinematics. This joint trajectory is then tracked by a compliant feed-forward controller, which also observes external perturbations. After trajectory execution, a reward, including those interaction forces and other feedback such as vision or task achievement is evaluated. Last, after a certain number of evaluated trajectories, a policy search algorithm updates the policy parameters.

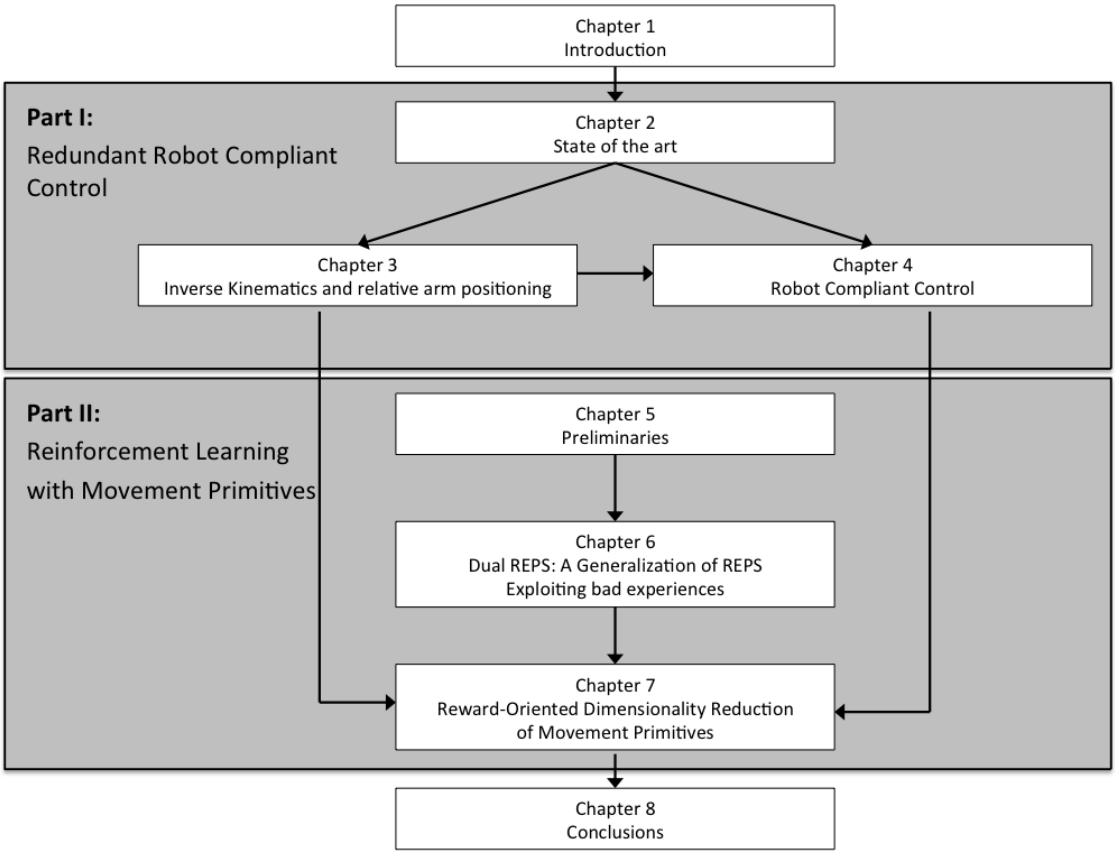


Figure 1.3: Thesis overview.

Part I

The first part is devoted to kinematics, workspace manipulability maximization and control.

- **Chapter** 2 provides a brief review of the work over the last decades on Inverse Kinematics for redundant robots, as well as on their relative positioning. Additionally, some of the approaches to compliant robot control and external force estimation are presented.

- **Chapter** 3 presents an analysis of the problems that arise when using CLIK algorithms. Then, an experimental comparison of such algorithms and two enhancements are presented to obtain a more robust inverse kinematics algorithm. Using such IK algorithm, we propose, for redundant robots, a method for optimizing the relative positioning of two arms, given certain constraints.

- **Chapter** 4 presents alternative ways to building an analytical inverse dynamics model for a robotic arm. Such inverse dynamics models allow us to have an estimate of contact forces acting on a robot during a trajectory. Real-robot experiments show the effectiveness of the dynamic models built for compliant manipulation.

Part II

The second part of this thesis is focused on motion characterization and learning.

- **Chapter** 5 introduces direct Policy Search (PS) and the algorithms used throughout this thesis: REPS and PI2. Additionally, it presents the most common frameworks for representing parametrized motion: Dynamic Movement Primitives (DMPs) and Probabilistic Movement Primitives (ProMPs).

- **Chapter** 6 proposes a generalization of REPS exploiting low-performance trajectory samples named Dual REPS (DREPS). This approach is applied to an illustrative 2-dimensional task, and also to a real-robot scenario with two optimal solutions.

- **Chapter** 7 then presents reward-oriented Dimensionality Reduction (DR) algorithms for performing PS with movement primitives. In particular, we apply linear DR on DMPs and ProMPs. We then test such DR methods in simulated robotic tasks, as well as a bimanual robot task of folding a polo shirt.

- **Chapter** 8 concludes the thesis, and presents future challenges this subfield of robotics.

Appendices

- Appendix A presents a list of the publications associated to this thesis.
- Appendix B lists a series of links to additional experimental results.
- Appendix C lists the acronyms used throughout this thesis.
- Appendix D defines the most relevant variables used.

Part I

Compliant Redundant Robot Control

2
State of the Art

The recent trend of building more human-like robots, as well as using them in uncontrolled environments such as households, presents several challenges that increase the complexity of task learning. It is then required to have an integrated system capable of properly handling the kinematics and dynamics of the robot in the process of learning. Additionally, bimanual robots need to be configured in a way their arms are able to coordinate their motion while handling an object. In the early stages of this PhD work, we realized some key elements were not developed enough in literature and therefore needed to be studied in depth and documented. Such elements are:

- **Robot kinematics**, specially Inverse Kinematics (IK) for redundant serial robots. While IK is a problem considered to be relatively solved, the robustness of some of the approaches proposed in literature is less than expected, due to kinematically singular robot configurations and limited joint values for each articulation. This causes robots to crash or have dangerous behaviors when using such kinematics algorithms without a planning framework ensuring such safety.

- **Bimanual manipulation**, in particular, the relative positioning of two robotic arms. Bimanual manipulation and workspace characterization have been broadly studied from several perspectives, but the literature on bimanual robot design is not as broad. Thus, given two serial manipulators and proper inverse kinematics algorithms, a study of the most appropriate relative positioning of two arms for a bimanual manipulation task was needed.

- **Compliant control and wrench estimation**. Additionally to the kinematics aspect, we wanted to have a safe robot behavior also capable of detecting external disturbances while performing tasks. Therefore, a dynamic model of the available robot needs to be built and used together with a disturbance observer in order to achieve such desired control. Such

disturbance signal becomes also helpful from the learning perspective, forming part of a reward function.

In this chapter, we provide an overview of the key theoretical elements previous to our work, in order to accomplish the kinematic and dynamic control, taking into account workspace capabilities, kinematics and control.

2.1 Inverse kinematics of redundant serial robots

Robot kinematics applies geometry to the study of the movement of multi-degree of freedom kinematic chains that form the structure of robotic systems [4]. The kinematics equations establish the relation between the robot articulations -joints- and the end-effector pose in the robot's operational space.

Forward Kinematics (FK) is a function that maps the robot's joints into a pose in the operational space. In the scope of this thesis, the forward kinematics equations of a serial robot always provide a unique solution by direct substitution of the joint values into the kinematics equations [4].

Inverse Kinematics (IK) is the inverse mapping of the FK function and, given a robot end-effector pose, provides a joint configuration with which the end-effector will be at the desired pose. If the number of Degrees of Freedom (DoF) or articulations of the robot, d, is smaller than the operational space dimension n, a robot is considered to be redundant, meaning that the IK mapping can have infinite many solutions for a certain desired pose. Redundancy provides robots with more flexibility, while it makes the IK computation more difficult.

Moving redundant robot arms in task space requires efficient and well-behaved Inverse Kinematics (IK) solutions. Along several decades, a lot of effort within the Robotics community has been devoted to obtaining fast and robust IK algorithms. Analytical methods have always been preferred to iterative ones, because their solution is exact and usually faster to compute. However, with the rise of redundancies in robots, analytical solutions become harder to obtain [28–30] and thus again alternatives need to be explored [31] [32] in order to benefit from the additional degrees of freedom [33]. In addition, complex tasks impose more restrictions on IK solutions, such as in the case of medical robots [34] [35].

Although many alternatives for trying to solve the IK problem exist, such as interval methods [36], distance-based methods [37], or even neural networks [38–40] and Bézier maps [41] [42], probably the most popular way is to use closed-loop algorithms, where a first-order geometric Jacobian matrix $\mathbf{J}$ [43] [44] of the robot kinematics is computed. Such geometric

Jacobian maps joint velocities $\dot{\mathbf{q}}$ into task space velocities $\dot{\mathbf{x}}$:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}. \tag{2.1}$$

Inverting the Jacobian matrix with a certain inverse operator $\star$,

$$\mathbf{J}^{\star}\dot{\mathbf{x}} = \dot{\mathbf{q}}, \tag{2.2}$$

and using finite differences, so that the task space error can be mapped into an update of the joint values that is likely to reduce the task space error, we obtain:

$$\Delta\mathbf{q} = \mathbf{J}^{\star}\mathbf{e}. \tag{2.3}$$

In an iterative procedure, given $\mathbf{q}^k$, the updated joint state at step $k+1$ is then $\mathbf{q}^{k+1} = \mathbf{q}^k + \Delta\mathbf{q}^k$, for some computed $\Delta\mathbf{q}^k$:

$$\Delta\mathbf{q}^k = \alpha\mathbf{J}^{\star}\mathbf{e}, \tag{2.4}$$

where α is a gain, $\mathbf{J}^{\star}$ is an inverse of the geometric Jacobian matrix and $\mathbf{e}$ is the task space positioning error.

The first attempts to close the IK loop used the Moore-Penrose pseudoinverse [45] of the Jacobian matrix [46] to invert the differential kinematics equation of the robot.

In other works, the Jacobian transpose was used [47], which is faster to compute. As a distinctive advantage over alternative IK methods, Closed-Loop Inverse Kinematics (CLIK) algorithms do not require any previous knowledge or learning process with the robot, other than its Jacobian matrix, this being the main reason for its preferred use over other options. However, CLIK algorithms become unstable when the robot is close to a singularity, i.e., certain robot configuration where the Jacobian matrix is ill-conditioned: the floating point error when inverting the Jacobian becomes very large, thus amplifying the numerical error at each iteration, and also requiring large variations in some joints in order to reduce the error in a given direction. To solve these problems, the Jacobian matrix can be damped or filtered [48, 49], reducing numerical error, but not always reducing large joint variations. Some attempts also use second-order derivatives of motion, i.e., calculating the Hessian matrix of the forward kinematics [50], although this requires much more computation time.

In a continuous time assumption, the convergence of closed-loop methods can be demonstrated in terms of Lyapunov theory [51, 52]. Nevertheless, real-time computations have a fixed step, lower bounded by the computation capability of a processor, thus convergence cannot always be ensured by means of Lyapunov theory. Although there exist discrete-time versions of

it [53], their application is not immediate, and some additional assumptions must be made.

There are also studies about the convergence of these methods which takes the discrete-time system as a sequence and proves its convergence. In [54], an upper bound on the gain α that guarantees convergence is found, but restricting the operational space to a subset where the Jacobian is full-rank with bounded singular values, so its application is not general. Nevertheless, this work points out the relevance of the initial error dependency for these methods to converge, so the closer to the goal, the better these methods perform in the initial steps. In general, a smaller step improves convergence rate on the one hand, but slows the algorithm on the other. But in fact, reducing the global gain is not a truly effective strategy to avoid large gains near a singularity, as we will be also damping the gain in the directions we would like the robot to move.

In [55] a Selective Damping (SD) of the gain on the joint variations derived from each task-space error component is proposed, that modifies each gain depending on the corresponding column of the Jacobian and a predefined maximum joint variation γ_{max}. This effectively solves the gain issue, but does not solve singularity issues, such as the loss of rank and algorithmic singularities.

Using first-order derivative algorithms of robot motion has also the drawback that, depending on the goal position, the robot can get stuck at an *algorithmic singularity*, a pose where the error $\mathbf{e}$ belongs to the kernel of the inverted Jacobian, or in a multiple-task algorithm, as we will see later, the joint variation derived from a secondary task takes the opposite value of that for the primary task, thus the total computed joint variation becomes $\Delta\mathbf{q} \simeq 0$.

The main advantage of redundancy is to be able to perform secondary tasks and/or to choose which solution suits some criterion best. To this purpose, an optimization function can be set to find, within the set of IK solutions, the one that performs best according to the criterion. The most common procedure is to project a gradient of a secondary task into the kernel of the Jacobian matrix, in order not to affect much the position error. Other algorithms like the Augmented Jacobian or the Extended Jacobian [56], in which rows are added to the Jacobian, have been used. Among the existing criteria for optimization, the *manipulability* measure [57, 58] is often used. Other criteria such as collision avoidance [4] (by setting a minimum distance to a certain object), minimum effort kinematics [59] or structural stiffness are also used [60]. But respecting joint limits is often the main priority when exploiting the redundancies of a robot.

Definition of error

For the position and orientation error representation as an n-dimensional vector, the incommensurability of position and orientation units is a limiting situation, as the Jacobian inverse is not invariant to rescalation and translation. This was a major problem at earlier stages of

hybrid control theory [61], which relied on an orthogonal complements structure that was not invariant wrt. rescaling of the units taken . One solution to this problem was to use metrics that converted all Jacobian components to energy units [62, 63]. However, when using a Jacobian matrix with disparate units for IK, despite it being dependent on the units taken and the relation between them, the convergence of CLIK algorithms is not affected. When using the orientation error in [4], the equivalence of $2rad = 1m$ is often taken and provides a reasonable error ratio between position and orientation.

Condition number

Given a system of the type $\Delta\mathbf{q} = J^{\star}\mathbf{e}$, where $\star$ denotes an inverse operator, it is very common to have numerical or measurement errors on the robot's task position, or uncertainty on the kinematic parameters of the robot [64] [65]. Therefore, we need to take into account some uncertainty $\delta\mathbf{e}$ on the position error $\mathbf{e}$ (difference between target and current positions). It is fundamental to avoid amplifying this uncertainty when computing $\Delta\mathbf{q}$. To this purpose, the relative error $\delta\mathbf{q}$ on $\Delta\mathbf{q}$ coming from the uncertainty $\delta\mathbf{e}$ on $\mathbf{e}$ can be estimated using the condition number of $J^{\star}$ [66, 67]:

$$\frac{\|\delta\mathbf{q}\|}{\|\Delta\mathbf{q}\|} \leq \kappa(\mathbf{J})\frac{\|\delta\mathbf{e}\|}{\|\mathbf{e}\|}, \tag{2.5}$$

where $\kappa(\mathbf{J}^{\star})$ is the Condition Number (CN) of $\mathbf{J}^{\star}$, computed as the ratio of its maximum and minimum singular values:

$$\kappa(\mathbf{J}^{\star}) = \frac{\sigma_{max}(\mathbf{J}^{\star})}{\sigma_{min}(\mathbf{J}^{\star})} \tag{2.6}$$

Note that the dependency of the Jacobian on the units taken and the orientation-translation equivalence chosen may affect the CN of the linear system solved to obtain $\Delta\mathbf{q}$. Nevertheless, the CN will only grow to infinity when the Jacobian is not full-rank, which happens when approaching a singularity. As singularities are invariant to rescalation, we can conclude that the error propagation when solving the linear system for joint increments will have the same behavior for any scale chosen for angles and distances.

Convergence and singularities

Regarding Equation (2.4), an appropiate inverse of the Jacobian matrix must be applied to make the algorithm converge to zero error. The Jacobian Pseudoinverse (JP) algorithm is widely used, as previously mentioned, it being a generalized inverse for non-square matrices that can be

defined as $\mathbf{J}^{\dagger} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$ when $\mathbf{J}$ is full-rank. With this inverse, the JP update rule is:

$$\Delta\mathbf{q} = \mathbf{J}^{\dagger}\mathbf{e}, \tag{2.7}$$

$\mathbf{e}$ being the task space positioning error.

When a robot reaches a singularity, the algorithm might get stuck in a point or the pseudoinverse matrix may have too large values. By computing the Singular Value Decomposition (SVD) of $\mathbf{J}$:

$$\mathbf{J} = U\Sigma V^T, \tag{2.8}$$

and taking the singular values of J, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$:

$\mathbf{J} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$, with $\mathbf{u}_i$ and $\mathbf{v}_i$ being the ith columns of U and V, respectively, and $r = \max\{i \mid \sigma_i > 0\}$.

As U and V are orthonormal matrices, the pseudoinverse of $\mathbf{J}$ is:

$$\mathbf{J}^{\dagger} = V\Sigma^{\dagger}U^T = \sum_{i=1}^{r} \frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^T. \tag{2.9}$$

This expression shows that when the robot gets close to a singularity (one for which σ_i becomes very small), very large gains occur when computing $\Delta\mathbf{q} = J^{\dagger}\mathbf{e}$. In addition, the CN of the pseudoinverse is $\kappa(\mathbf{J}^{\dagger}) = \frac{\sigma_1}{\sigma_n}$, which tends to infinity at a singularity, thus losing numerical precision.

In Chapter 3, we provide a comparative overview of the different CLIK algorithms found in literature, also regarding numerical error propagation, which is sometimes forgotten. When analyzing these algorithms, the step gain α can be omitted, as it only affects the locality of convergence.

2.2 Bimanual manipulation

Bimanual manipulation allows robots to perform more complex tasks than a single-limb robot. Smith et al. [68] define bimanual manipulation as a subtype of dual-arm manipulation, where both arms are coordinated so that they are performing the same action in the sense a single arm can't possibly do its part without the other arm.

While a lot of attention in literature [68] is focused on how to manipulate or plan a task, less importance is given to the arms configuration. Usually, a humanoid-like configuration is chosen, to make the robot more human-friendly [69], but when deciding how to use two robots in a collaborative manner, it is therefore obvious to question if a humanoid-like configuration would

be the best, for example, for folding clothes. A first step towards this aim is to analyze the robot's workspace. In [70], a discretized workspace is used with information about the probability of solving the Inverse Kinematics (IK) with random orientations at each Cartesian position, and also manipulability data, an indicator of dexterity and distance to a singularity [57], in order to decide the grasping points for bimanual manipulation. However, this work exploits an existing humanoid robot, which may not have been specifically designed for the task being tackled. Zacharias et al. [71–73] plot the 3D Cartesian position workspace by initially drawing spheres, whose color varies with the percentage of inverse kinematics solutions found for each point. Moreover, they propose to use different shapes at each point to represent orientations, depending on the feasible end-effector orientations at each position, but their later work also focuses on optimizing manipulation with a given bimanual robot, rather than deciding its arms configuration. In these terms, few studies about such relative positioning of arms have been made to improve the kinematics capabilities of a bimanual robot.

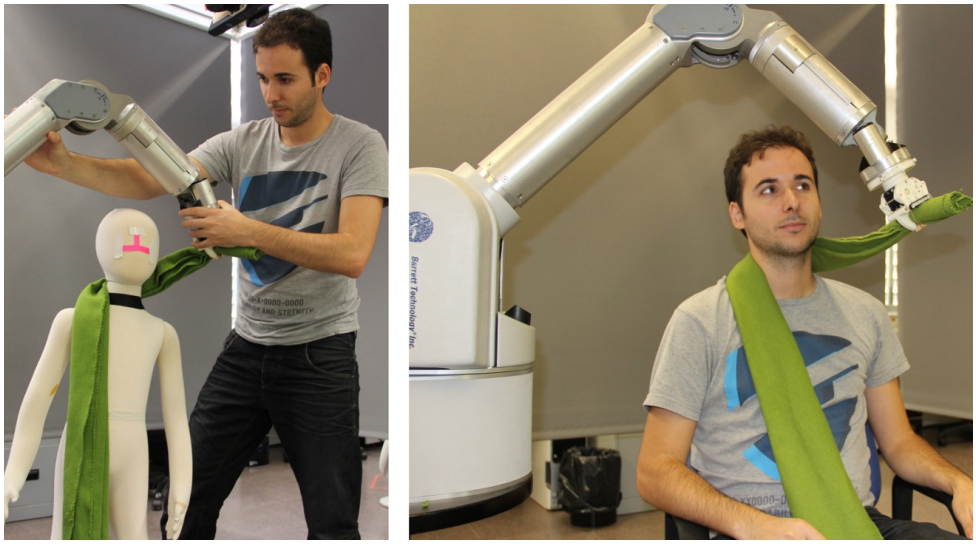


Figure 2.1: Kinesthetic teaching of a task to a WAM robot (left) and motion reproduction with a compliant controller (right).

2.3 Compliant control and wrench estimation

The interest in human-robot interaction is growing nowadays, thanks to the increasing availability of more compliant robots with low inertia, making them safer to move in a soft or fragile environment, in which we could include interaction with humans. Tasks like placing a scarf on a mannequin (see Fig. 2.1) can be taught to the robot by using Dynamic Movement Primitives

(DMP) and then reproduced with high precision. However, in compliant environments, there is usually a tradeoff between precision and safety, since moving the robot with more precision (commonly with a high error-compensating term) will make its motion *stiff*, which makes it dangerous for a human. The robotics research community is actively working on generating solutions to realize robotics abilities to support daily life domestic tasks [74–76], such as manipulating cloth (see Fig. 2.2). For this reason, model-based controllers such as Computed Torque Control (CTC) [77] can be used, for which we need an Inverse Dynamic Model (IDM) of the robot, i.e., the mapping from the position, velocity and acceleration to the total torques acting on the robot. The IDM of a robot can be computed by derivating its mechanical energy, or by iterative methods (see [4], Chapter 7), or fitted by a regression model from real execution data, such as Locally Weighted Projection Regression (LWPR) [78].

Robots able to safely interact with their surroundings should have structural features like lightweight links and coupled joints actuators mechanisms [16] enabling them to perform compliant motions. Besides these, their low-level control architecture should avoid excessive stiffness, usually imposed by accuracy demands, as mentioned above.

Another major ingredient for the achievement of compliant robot behaviors is the need to supervise the external forces (and torques) generated along the robot motions. External forces may play diverse roles during the planning and execution of compliant robot motions. For instance, in force control schemes, the external manipulator wrench $\mathbf{f_e} \in \mathbb{R}^6$ is compared to a reference signal in order to have a desired end-effector interaction with the environment. Other schemes such as compliant control, impedance control or hybrid control also use the external wrench data to compute the corresponding system action [4].

For the purpose of making available the external wrench felt by a robot manipulator, expensive sensors are often used. In order to avoid the use of such devices, recent works [79] [80] present approaches for estimating the wrench or, at least, the joint torques due to an external action during manipulation.

However, most of the current approaches are based on the availability of an accurate analytical model of the robot dynamics, which may lead to inaccuracy due to modeling errors. This is specially true in modern robotics systems that are highly non-linear and can no longer be accurately modeled using the rigid body dynamics. In the specific case of the Barrett WAM [16], the analytical dynamic model becomes harder and much more complex to obtain for structural reasons, given that several spinning drives, some of them coupled, are in different reference frames than the joints they actuate with only one being measurable, resulting in effects such as reflective inertias.

Moreover, in a lightweight robot, any small error in the dynamic parameters like the link masses represents a large percentage error for the model accuracy. Interestingly, those structural

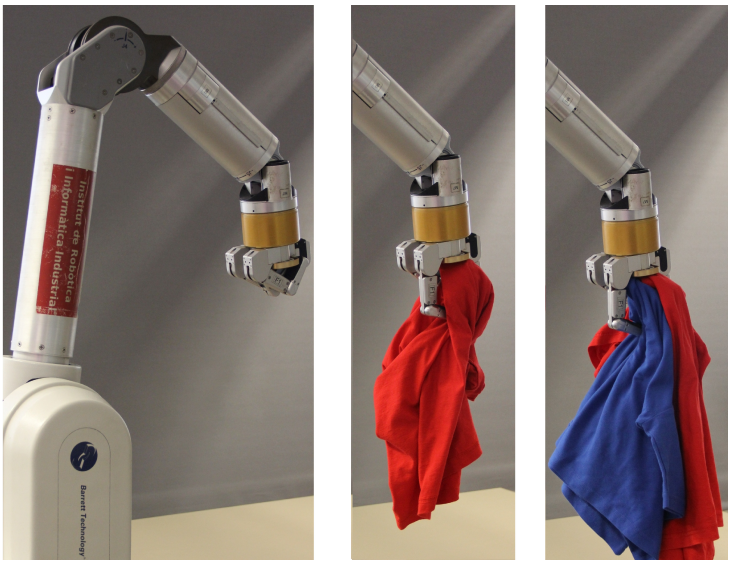


Figure 2.2: 7-DoF WAM robot holding none, one and two cloth garments. A proper external force estimation would help the robot to know how many garments have been picked after an action.

features that allow a robot to be compliant make it harder to model and, therefore, estimation of contact forces using state-of-the-art methods is more difficult, which conversely imposes restrictions on the exploitation of the physical compliance capability of the robot.

Modern methods for wrench estimation are based on the use of state space observers [79] [80]. The rationale behind this idea is that the robot is experiencing external forces that produce changes in its state, therefore, by estimating the internal state of the system and assuming that a certain part of the total inputs is known, an estimation of perturbations (external inputs/forces) can be completed. As described in Section 4.1.1, such observers are based on the availability of the analytical model of the manipulator dynamics.

2.3.1 Learning robot inverse dynamics

The dynamics of a serial robot, as described in [4], is given by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F_f}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{u_T}, \tag{2.10}$$

where $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^n$ denote joint angles, velocities and accelerations of the robot with a number d of Degrees of Freedom (DoF), $\mathbf{M}(\mathbf{q})$ represents the inertia matrix, and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, $\mathbf{G}(\mathbf{q})$ and $\mathbf{F_f}(\mathbf{q})$ are the Coriolis and centripetal, gravity and friction forces acting on the robot. Finally, $\mathbf{u_T} \in \mathbb{R}^n$ is the vector of total input forces to the joints. We assume that such forces may proceed from

applied torque commands $\mathbf{u_c}$ and from certain external torque $\mathbf{u_e}$. Thus,

$$\mathbf{u_T} = \mathbf{u_c} - \mathbf{u_e}.$$

At the same time, the inverse model of the robot dynamics is a function mapping the robot state to the actions that would generate it, which in the absence of external forces would be given by

$$\mathbf{u_c} = \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}). \tag{2.11}$$

To obtain this function $\mathbf{g}$, model learning is a very active research field in robot control [77] where methods are developed allowing the approximation of (2.11) using input/output data. Online model learning includes methods like Locally Weighted Projection Regression (LWPR) [78], Local Gaussian Process (LGP) [77] or Gaussian Mixture Models (GMM) [81]. These approaches allow to improve the model even when the system is in operation.

Assuming that the function $\mathbf{g}$ has been learned, it can be stated that, given a dynamic state produced by both control and external torques, the inverse model would provide,

$$\mathbf{u_T} = \mathbf{u_c} - \mathbf{u_e} = \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}),$$

and as the vector $\mathbf{u_c}$ is assumed to be known, the estimation of the external torque would be straightforward.

However, there are a number of practical considerations when using a state observer for the external wrench estimation. The following two are considered the most relevant for this work:

Local learning vs. global learning

In the case of a 7-DoF robot such as the WAM robot, learning a function that maps a joint position, velocity and acceleration to a torque vector means a 21-dimensional input and a 7-dimensional output. This high dimensionality makes global learning difficult to achieve, as it would generate a large number of kernel functions to evaluate when using the model, and so a slow computation rate. In addition, various unmodelled friction factors such as static/dynamic friction, motor cogging and others cause different residual friction in the same position, depending on the trajectory followed.

Learning with and without accelerations

Measurements obtained for the WAM arm are joint positions, velocities (obtained by differentiating positions), and accelerations (as a second derivative). These derivatives are very sensitive to noise, making the simple approximation unsuitable.

A very small noise in a joint position measurement results in a very large noise in acceleration measures. Even with the use of heavy filters, such as Parks-McClellan filters [82], which minimize error in pass-and-stop bands and are used here to damp frequencies representing high acceleration in joints, the noise could not be completely mitigated to have a good dataset for learning a task. In order to overcome this problem, in this work we decided to use the trajectory given by joint position $\mathbf{q}$, velocity $\dot{\mathbf{q}}$, and acceleration $\ddot{\mathbf{q}}$ of the robot when learning a task and, instead of learning the whole dynamic system, we propose to learn the function:

$$\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{u_c} - \mathbf{M}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F_f}(\mathbf{q}, \dot{\mathbf{q}}),$$

that is, assuming that the only parameter of the robot to be known is the inertia matrix $\mathbf{M}(\cdot)$,

$$\mathbf{u_c} - \mathbf{M}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} = \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}).$$

This function, $\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}})$, only depends on the joint positions and velocities which allow for a more accurate learning. Figure 2.3 presents an example of the data used to learn this relation.

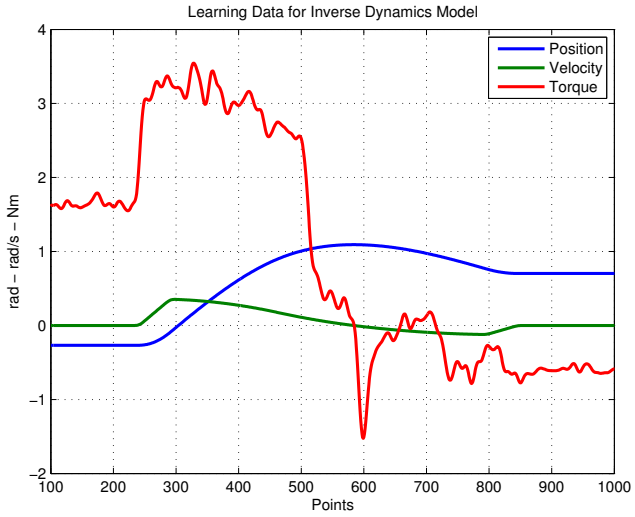


Figure 2.3: Data used for learning a trajectory, using position and velocity as inputs and torque as output.

2.3.2 Fitting an analytical model

While the mentioned methods show acceptable performance when running a controller with an inverse dynamic model, none of these methods takes into account the impact of high hysteresis on the dynamics of some robots, as it occurs with the WAM robot, where for non-high speed

motion the friction is usually the second highest torque acting on the robot after gravity. In such robot, the inertia, Coriolis and gravity terms can be analytically modeled [4], but the friction term needs a particular model.

Building a friction model

As it has been already mentioned, in the absence of external forces, we can model/compute the inertia, Coriolis, centripetal and gravity forces, and the controller torques are assumed to be known, thus we can infer the values of friction as:

$$\mathbf{F}_f = \mathbf{u}_c - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{C}(\dot{\mathbf{q}}, \mathbf{q}) - \mathbf{G}(\mathbf{q}), \tag{2.12}$$

and use them to fit a friction model. To obtain the terms in (2.12) in the case of the WAM robot, the code provided by the manufacturer already included an application to calibrate the gravity values, while geometry, masses and inertia terms were also provided by the manufacturer and can be used to obtain the Coriolis, centripetal and inertia terms.

Basic friction model

As mentioned, the friction torque applied to any joint of the robot when it moves at low speed is usually the second highest torque acting on the robot after gravity. This friction has a high hysteresis in its dynamic behavior which makes it very difficult to model. The reason to fit the friction as an hysteresis function, rather than with a *complete* set of basis functions, is that we know the qualitative behavior of the friction. Thus, using a proper fitting function will be more efficient than using any other type of kernel in terms of precision, number of parameters, and samples required for the fitting process.

At first, this torque was modeled as a viscous friction $F_f = c\dot{q}$, where $\dot{q}$ is the joint's velocity and c is a constant that varies depending on the joint that moves. This model was not very precise and did not model the friction with a high degree of accuracy because some other variables had not been considered, such as the position of the joints.

In [83], the friction is modeled with an initial model as:

$$F_f^i = b_1 atan(s\dot{q}_i) + b_2\dot{q}_i, \tag{2.13}$$

where $\dot{q}_i$, is the ith joint velocity and b_1, b_2 and s are parameters obtained with least-squares techniques.

However, this model not only is independent from position (which we observed as a fact from data), but also has no hysteresis value (friction is zero for zero velocity). Thus we added

a term to model this hysteresis with a parameter z defining its amplitude and the sign of the acceleration of the joint, leads to our basic hysteresis model:

$$F_f^i = \quad b_1 atan(s\dot{q}_i + z sign(\ddot{q}_i)) + b_2.\dot{q}_i, \text{ for i} = 1..7 \tag{2.14}$$

This basic model did not offer the level of accuracy required because it did not show the linear dependence the friction could have wrt. the position of the joints as seen in Section 4.2 of this thesis.

2.4 Summary

In this chapter, we provided the existing basis for a better understanding of the work presented in the first part of the thesis. We introduced the Closed-Loop Inverse Kinematics (CLIK) algorithms in Section 2.1, defining the basic update rule and key elements to be used in Chapter 3 such as the robot Jacobian, the pseudoinverse matrix, the singular value decomposition or the condition number. Moreover, we introduce the literature regarding bimanual workspace analysis in Section 2.2, and Section 2.3 introduces the dynamics equation of a robot, as well as the problematics of obtaining a reliable inverse dynamic model.

In Chapter 3, we analyze the performance of most of the existing CLIK algorithms with their associated conditioning and convergence, and present two improvements resulting in a more stable IK algorithm, which can then be used for converting robot Cartesian space commands into joint space commands.

Chapter 4 proposes a stable wrench estimator when using a compliant controller. In particular, feed-forward controllers are used. Section 4.2 also presents an improvement of the dynamic models needed for such feed-forward controllers for the specific case of the Barrett WAM robot.

3

Inverse Kinematics and Relative Arm Positioning

This chapter proposes two enhancements to the current state-of-the-art Closed-Loop Inverse Kinematics (CLIK) algorithms in Section 3.1, to then apply them to analyse and assess the relative positioning of two arms for cooperative manipulation in Section3.2.

3.1 Inverse kinematics of redundant robots

In tuning the Inverse Kinematics (IK) of the 7-DoF WAM manipulator to the particular requirements of some applications, we noticed that the existing generic KDL algorithm [84] could sometimes fail due to joint limits. We tried other open-source IK algorithms [85], but none performed to entire satisfaction, thus we explored other possibilities for redundant IK.

Focusing on solving the IK with feasible joint values, two enhancements upon the state-of-the-art are proposed. The first one is a way of filtering the Jacobian matrix that ensures a given numerical conditioning, while the second uses the advantages of the latest works on continuity of inverse operators applied to robotics [86] with a controlled step size [55] to smoothen the motion of the robot. All the analyzed algorithms, as well as the proposed enhancements, have been implemented on a Barrett WAM robot and tested both in simulation and in real experimentation.

For the methods presented, we will be using the abbreviations in Table 3.1.

3.1.1 CLIK algorithm issues

In this section, we present a comparative overview of the different state-of-the-art CLIK methods present in literature, and the problematics of each of them regarding convergence and numerical stability. As stated in Chapter 2, CLIK methods are based on inverting the robot's geometric Jacobian in order to map errors to joint changes, as in Eq. 2.4.

Table 3.1: Methods abbreviations

Name	Abbreviation	Equation/Section
Jacobian Pseudoinverse	JP	(2.7)
Jacobian Transpose	JT	(3.1)
Selective Damping	SD	Section 3.1.1
Jacobian Damping	JD	(3.3)
Jacobian Filtering	JF	(3.6)
Error Damping	ED	(3.7)
Improved Error Damping	IED	(3.8)
Singular Value Filtering	SVF	Section 3.1.5
Jacobian Weighting	JW	(3.16)
Gradient Projection	GP	(3.17)
Joint Clamping	JC	(3.20)
Task Augmentation	TA	Section 3.1.3
Task Priority	TP	(3.25)
Continuous Task Priority	CTP	(3.26)

Jacobian transpose

To gain computation speed, the Jacobian Transpose (JT) method uses, instead of an inverse of the matrix $\mathbf{J}$, its transpose with the following control rule [87]:

$$\Delta\mathbf{q} = \mathbf{J}^T\mathbf{e}, \tag{3.1}$$

where $\mathbf{J}^T$ is now the transpose of the geometric Jacobian of the manipulator. This method has a computationally very fast step, although it may require more steps than other methods, and not being a least-squares solution can derive in chattering.

Following, other alternatives to Eq. (2.4) are described, to reduce both the gain magnitude or large conditioning on the matrix inversion.

Selectively damped pseudoinverse (SD)

As previously seen, a small singular value in the Jacobian yields large gains in all directions. In [55], a new way of controlling the step magnitude is defined, which consists in damping differently the effect of each one of the components of the position error, expressed in the basis of the singular value decomposition of $\mathbf{J}$. Hence small singular values of the Jacobian, which would turn into large gains, are damped more severely.

In [55], a unitary error in the direction of one of the eigenvectors in the task space (columns of U) is taken, $\mathbf{e} = \mathbf{u}_i$, and a bound on the joint variation associated to this error component

is obtained, which is then used to damp the gain of the corresponding error component and, finally, the gains over all the error components are added up. This results in a $\Delta\mathbf{q}$ with limited gains at each component of the task space.

Jacobian damping (JD)

To avoid the discontinuity of the JP operator which results in a large conditioning of the Jacobian, meaning numerical error, the Jacobian Damping or singularity robust pseudoinverse was proposed [88] [89] [48] [49] [60]. If we rewrite the Jacobian matrix, using its singular value decomposition, as:

$$\mathbf{J}_D = \sum_{i=1}^{d} \frac{\sigma_i^2 + \lambda^2}{\sigma_i} \mathbf{u}_i \mathbf{v}_i^T, \tag{3.2}$$

then the Jacobian Damping algorithm will use the following pseudoinverse:

$$\mathbf{J}^{\dagger_D} = \mathbf{J}_D^{\dagger} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2 I)^{-1} = \sum_{i=1}^{d} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \mathbf{v}_i \mathbf{u}_i^T, \tag{3.3}$$

which, for some small λ (usually around 10^{-3}), is almost the same matrix as the ordinary pseudoinverse when $\sigma_i^2 \gg \lambda^2 \ \forall i < d$, and when the smallest singular value σ_d is close to zero, $\lim_{\sigma_d \to 0} \frac{\sigma_d}{\sigma_d^2+\lambda^2} = 0$, instead of ∞.

The JD algorithm avoids discontinuities in the Jacobian's singular values, and it provides the solution minimizing $\|\mathbf{J}\Delta\mathbf{q} - \mathbf{e}\| + \lambda^2\|\Delta\mathbf{q}\|$, which will result in a smaller norm solution. However, a small λ value does not guarantee that $\|\Delta\mathbf{q}\|$ will be small and an analysis of the CN shows it provides no guarantee of keeping the numerical error within an acceptable range.

Let $g(\sigma) = \frac{\sigma}{\sigma^2+\lambda^2}$ be the function used instead of a trivial inversion $1/\sigma$ when computing the pseudoinverse of the Jacobian as in Eq. (2.9). This function g, as we see in Fig. 3.1, has a maximum at $\sigma = \lambda$ with a value of $g(\lambda) = \frac{1}{2\lambda}$.

Now, considering the problem of solving $\mathbf{e} = \mathbf{J}_D\Delta\mathbf{q}$, we can distinguish several cases depending on the least singular value (σ_d) of the geometric Jacobian:

- $\sigma_d > \lambda$, or $\frac{\lambda^2}{\sigma_i} < \sigma_d < \lambda$, $\forall i \neq d$. Then $g(\sigma_d) > g(\sigma_i)$, $\forall i \neq n$ and the condition number is:

$$\kappa(\mathbf{J}^{\dagger_D}) = \frac{\sigma_d(\sigma_1^2 + \lambda^2)}{\sigma_1(\sigma_d^2 + \lambda^2)} \xrightarrow{\sigma_d \to \lambda} \frac{\lambda^2 + \sigma_1^2}{2\sigma_1\lambda} \in O\left(\frac{1}{\lambda}\right) \tag{3.4}$$

- $\exists i, j$ so that $\frac{\lambda^2}{\sigma_i} < \sigma_d < \frac{\lambda^2}{\sigma_j}$. Then we have $g(\sigma_i) < g(\sigma_d) < g(\sigma_j)$ and, as the condition number bound will not depend on σ_d, it will be bounded.

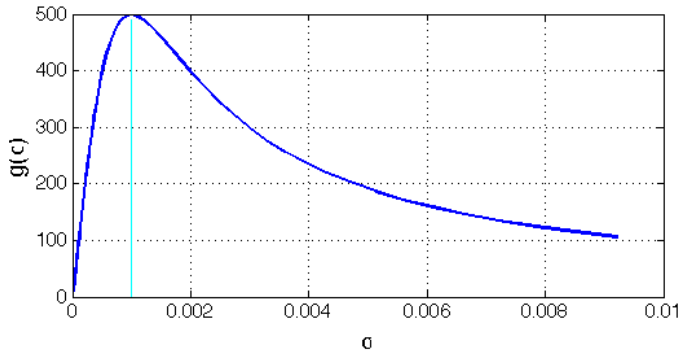


Figure 3.1: Inverse of a singular value for the Jacobian Damping algorithm for $\lambda = 10^{-3}$. Note its maximum at $g(\lambda) = \frac{1}{2\lambda}$.

- $\sigma_d < \frac{\lambda^2}{\sigma_i}, \forall i \neq d$. Then $g(\sigma_d) < g(\sigma_i)$ and we now have (for some k):

$$\kappa(\mathbf{J}^{\dagger_D}) = \frac{\sigma_k(\sigma_d^2 + \lambda^2)}{\sigma_d(\sigma_k^2 + \lambda^2)} \overset{\sigma_d \to 0}{\longrightarrow} \infty \tag{3.5}$$

This means that, on the one hand, λ should have a high value to avoid the maximum $1/(2\lambda)$ of the condition number at $\sigma_d = \lambda$, but on the other hand, λ must also have a very small value to avoid entering the last case, in which the conditioning tends to infinity as σ_d decreases.

As it is not always necessary to damp the Jacobian, and in many cases the necessary damping may vary, a singular region can be defined so that damping is applied only when entering it. To this purpose, a variable damping factor, leading to Jacobian Filtering (JF) can be used as in [49]:

$$\lambda^2 = \begin{cases} 0 & \text{if } \sigma_d \geq \epsilon \\ (1 - (\frac{\sigma_d}{\epsilon})^2)\lambda_{max}^2 & \text{if } \sigma_d < \epsilon \end{cases} \tag{3.6}$$

where σ_d is the smallest singular value of the Jacobian matrix, ϵ is the width of a singular region (in terms of singular values) in which the damping factor takes a non-zero value, and λ_{max} is the maximum damping factor allowed.

When using the JF algorithm, the function g becomes $g_F(\sigma) = \frac{\sigma}{\alpha\sigma^2 + \lambda^2}$, with $\alpha = 1 - (1/\epsilon^2)$, so the CN behavior does not change wrt. the JD algorithm.

Error damping (ED)

Another option for damping the pseudoinverse matrix is to use the current positioning error. In this way, Chan and Lawrence [90] proposed an Error Damped (ED) pseudoinverse matrix

defined as:

$$\mathbf{J}^{\dagger ED} = \mathbf{J}^T \left(\mathbf{J}\mathbf{J}^T + EI_m\right)^{-1} = \sum_{i=1}^{d} \frac{\sigma_i}{\sigma_i^2 + E}\mathbf{v}_i\mathbf{u}_i^T, \tag{3.7}$$

where $E = 1/2\mathbf{e}^T\mathbf{e}$. Using this damping term strongly reduces the gains when far from the goal, but if $\sigma_d < \frac{E}{\sigma_i}$, $\forall i$, then $\kappa(\mathbf{J}^{\dagger ED}) \in O\left(\frac{1}{\sigma_d}\right)$ which can still become a large conditioning.

Equation (3.7) may have small singular values when the error is small. In such case, the condition number would rise again. To avoid this situation, in [91], an improved version of the error-damped pseudoinverse (IED) is proposed by adding a term $\omega I_m = diag(\omega_1, ..., \omega_m)$, being $\omega_i \simeq 10^{-1}l^2 \sim 10^{-3}l^2$, with l the characteristic length of the links [91]:

$$\mathbf{J}^{\dagger IED} = \mathbf{J}^T \left(\mathbf{J}\mathbf{J}^T + EI_m + \omega I_m\right)^{-1} = \sum_{j=1}^{d} \frac{\sigma_i}{\sigma_i^2 + E + \omega_i}\mathbf{v}_i\mathbf{u}_i^T. \tag{3.8}$$

This last proposal (3.8) is more robust than the filtering/damping methods, but when the goal position is singular, the inversion behaves in the same way and it may suffer conditioning issues as all the other filtering algorithms.

3.1.2 First enhancement: Singular Value Filtering (SVF)

In order to overcome the conditioning problems of the JD, we propose to modify the Jacobian matrix' singular values so that it never loses rank and its condition number is bounded. To this purpose, if we take the SVD of J:

$$\mathbf{J} = U\Sigma V^T = \sum_{i=1}^{m} \sigma_i\mathbf{u}_i\mathbf{v}_i^T, \tag{3.9}$$

then we define

$$\hat{\mathbf{J}} = \sum_{i=1}^{m} h_{\nu,\sigma_0}(\sigma_i)\mathbf{u}_i\mathbf{v}_i^T, \tag{3.10}$$

where

$$h_{\nu,\sigma_0}(\sigma) = \frac{\sigma^3 + \nu\sigma^2 + 2\sigma + 2\sigma_0}{\sigma^2 + \nu\sigma + 2} \tag{3.11}$$

is our proposed filtering rational function, where σ_0 is the minimum value we want to impose to the singular values of $\mathbf{J}$, and ν is a shape factor, that regulates the curvature (shape) of function $h_{\nu,\sigma_0}(\sigma)$.

With Eq. (3.11) we can compute (assuming $\sigma_i > \sigma_{i+1}$ $\forall i$)

$$\hat{\mathbf{J}}^{\dagger} = \sum_{i=1}^{d} \frac{1}{h_{\nu,\sigma_0}(\sigma_i)}\mathbf{v}_i\mathbf{u}_i^T, \tag{3.12}$$

to use it as the pseudoinverse. In this expression, it can be easily seen that $h_{\nu,\sigma_0}(\sigma)$ verifies:

- $h_{\nu,\sigma_0}(\sigma)$ is continuous and differentiable on the positive side of $\mathbb{R}$ which is where the singular values are.
- $\lim_{\sigma\to 0} h_{\nu,\sigma_0}(\sigma) = \sigma_0$, $\forall\nu$, thus σ_0 is the minimum value we will allow for the singular values of the Jacobian matrix.
- $h_{\nu,\sigma_0}(\sigma)$ has an asymptote of equation $y = \sigma$ for $\sigma \to \infty$, as $\lim_{\sigma\to\infty} \frac{h_{\nu,\sigma_0}(\sigma)}{\sigma} = 1$ and $\lim_{\sigma\to\infty} (h_{\nu,\sigma_0}(\sigma) - \sigma) = 0$, $\forall\nu$ and $\forall\sigma_0$.
- $h_{\nu,\sigma_0}(\sigma)$ is monotonic wrt. σ if ν and σ_0 are defined verifying $\nu > \sigma_0$ and $2 > \nu\sigma_0$, which are not very restrictive conditions. This monotonicity guarantees that the condition number of the pseudoinverse (3.12) is always:

$$\kappa(\hat{\mathbf{J}}^\dagger) = \frac{(\sigma_1^3 + \nu\sigma_1^2 + 2\sigma_1 + 2\sigma_0)(\sigma_d^2 + \nu\sigma_d + 2)}{(\sigma_1^2 + \nu\sigma_1 + 2)(\sigma_d^3 + \nu\sigma_d^2 + 2\sigma_d + 2\sigma_0)}. \tag{3.13}$$

  Therefore we have:

$$\lim_{\sigma_d\to 0} \kappa(\hat{\mathbf{J}}^\dagger) = \frac{(\sigma_1^3 + \nu\sigma_1^2 + 2\sigma_1 + 2\sigma_0)}{\sigma_0(\sigma_1^2 + \nu\sigma_1 + 2)} = \frac{A(\sigma_1)}{\sigma_0}, \tag{3.14}$$

  which is always bounded by the inverse of the minimum value assigned to the singular values.

To sum up, $\hat{\mathbf{J}}$ has lower-bounded singular values and tends to $\mathbf{J}$ when its singular values move away from 0. Moreover, with this filtering, the Jacobian matrix never loses rank as the singular values are strictly positive.

Figure 3.2 displays the condition number of different methods in the case of a 4R planar manipulator moving towards a singularity, for a damping factor of $\lambda = 10^{-2}$, and allowing a maximum damping factor on the filtering algorithm (variable damping factor) of $\lambda_{max} = 5\lambda$. The results, plotted in logarithmic scale, show that all previous filtering methods fail, at some point, at keeping the CN stable, while our proposal, with $\sigma_0 = 0.005$, $\nu = 10$, presents a bounded conditioning.

As a first test, these filtering algorithms have been applied to an example trajectory with a 4R manipulator, as recorded in Fig. 3.3, where it is clear that, while the JP and the JF have very large gains, and the JT and the IED are easily stuck and still unable to solve the gain issue, the SD performs smoothly, and our proposal, combining a method to avoid large condition number (SVF) with a bound on the gains (SD) makes the robot move smoother than with the other methods presented up to this point.

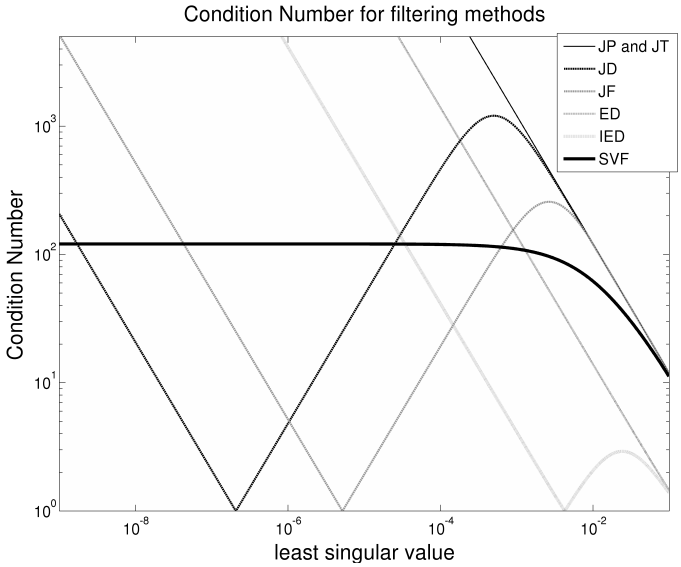


Figure 3.2: Condition Number, in logarithmic scale, for different methods on a 4R planar robot approaching a target singular position.

In addition, Table 3.2 shows the reprojection error on the task space for each algorithm, i.e., the computed difference on each eigenvector component between JP and the other alternatives. We can see that, the larger the parameter ν is, the smaller the reprojection error, thus a reasonably large number verifying $\sigma_0 < \nu < 2/\sigma_0$ is recommended (a value of 10 has been used throughout this chapter).

Table 3.2: Reprojection error.

Method	$\mathbf{J}^\star$	$\mathbf{J}(\mathbf{J}^\dagger\mathbf{e} - \mathbf{J}^\star\mathbf{e})$
JP	$\frac{1}{\sigma}$	0
JT	σ	$\sigma - 1$
JD	$\frac{\sigma}{\sigma^2+\lambda^2}$	$\frac{\lambda^2}{\sigma^2+\lambda^2}$
JF	$\frac{\sigma}{\sigma^2+\lambda_{max}^2(1-(\sigma/\epsilon)^2)}$	$\frac{\lambda_{max}^2\left(1-(1/\epsilon)^2\right)}{\sigma^2(1-(\sigma/\epsilon)^2)+\lambda_{max}^2}$
ED	$\frac{\sigma}{\sigma^2+0.5\cdot\mathbf{e}^T\mathbf{e}}$	$\frac{0.5\cdot\mathbf{e}^T\mathbf{e}}{\sigma^2+0.5\cdot\mathbf{e}^T\mathbf{e}}$
IED	$\frac{\sigma}{\sigma^2+0.5\mathbf{e}^T\mathbf{e}+\omega}$	$\frac{0.5\cdot\mathbf{e}^T\mathbf{e}+\omega}{\sigma^2+0.5\mathbf{e}^T\mathbf{e}+\omega}$
SVF	$\frac{\sigma^2+\nu\sigma+2}{\sigma^3+\nu\sigma^2+2\sigma+2\sigma_0}$	$\frac{\sigma_0}{\sigma^3+\nu\sigma^2+2\sigma+2\sigma_0}$

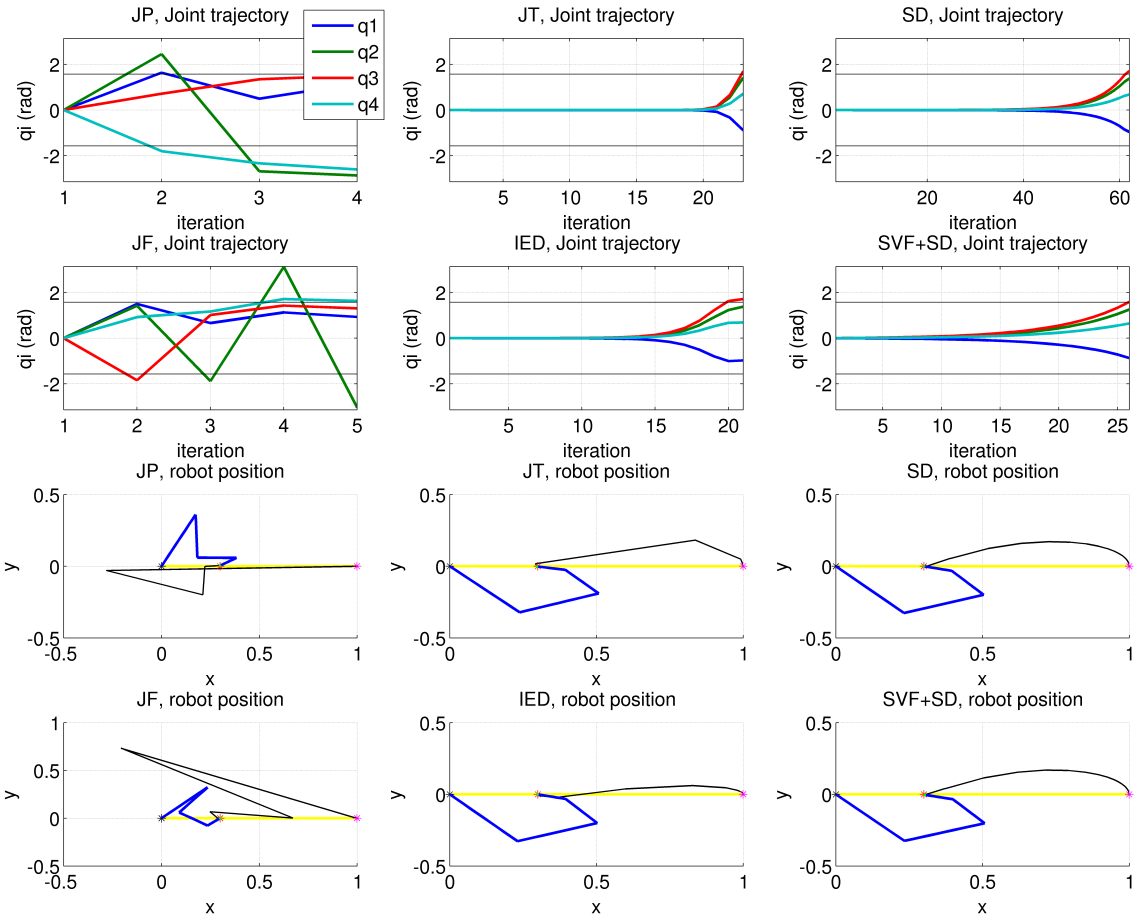


Figure 3.3: Behavior of some CLIK algorithms (see Table 7.1 for notation) applied to a 4R planar manipulator. The upper plots show joint trajectories, with joint limits marked with horizontal lines. Note the different scales for the x-axis. The lower plots show the robot frame, evolving from a light color pose at the start position to a darker one, with its end-effector's trajectory marked in a thiner black line.

3.1.3 Multiple tasks

Usually, when computing the IK of a robot, it is a good idea to compute not just a solution of the inverse kinematics, but the solution that behaves best according to a certain criterion. Even more with redundant robots, where the number of solutions may be infinite.

Jacobian weighting (JW)

Apart from considering metrics on the task space, metrics on the joints space can also be used. This metric (which can be variable) can be used to achieve secondary goals, prioritize the motion of certain joints, or even block a joint. In [92], Chan and Dubey defined the $\Delta\mathbf{q}$ that minimizes

$$\|\Delta\mathbf{q}\|_W = \sqrt{\Delta\mathbf{q}^T W \Delta\mathbf{q}}, \tag{3.15}$$

where $W$is an $m \times m$ diagonal matrix applying a weight to each joint depending on its relevance (according to a specified criterion), instead of the common Euclidean norm. Taking $\Delta\mathbf{q}_W = \mathbf{J}_W^\dagger \Delta x$, then

$$\Delta\mathbf{q} = W^{-1}\mathbf{J}^T(\mathbf{J}W^{-1}J^T)^{-1}\mathbf{e}, \tag{3.16}$$

where the influence of $w_i = W_{i,i}$ is that, the greater the w_i, the less q_i will vary in the given step.

Gradient projection (GP)

Another optimization option is to create a secondary task as a gradient of a function, and project it to the kernel of the primary task.

Given a cost function H, one can calculate its gradient ∇H, and project it onto the kernel of the matrix $\mathbf{J}$. Knowing that for any position of the manipulator, $\delta\mathbf{x} = \mathbf{J}\delta\mathbf{q}$. This means that if $\delta\mathbf{q} \in \ker(\mathbf{J})$, then $\delta\mathbf{e} = 0$, so the added term would not affect the error. In practice, as the step is not infinitely small, the linearization done by projecting the vector to the nullspace would indeed generate some additional error.

This projection onto the kernel is accomplished by multiplying any vector v by the matrix $P = I - \mathbf{J}^\dagger\mathbf{J}$.

Hence the GP is expressed as:

$$\Delta\mathbf{q} = \mathbf{J}^\dagger\mathbf{e} + \mu P \nabla H, \tag{3.17}$$

with μ a scalar indicating the magnitude of the projection, and ∇H the vector to project.

Task priority (TP)

The GP idea can be generalized to Task Priority (TP) algorithms [93], where an ordered set of tasks $t_1, ..., t_k$ is specified, their errors $\mathbf{e}_1, ...\mathbf{e}_k$ and Jacobians $\mathbf{J}_1, ...\mathbf{J}_k$ are computed, and next each task error is projected onto the kernel of the Jacobian of the previous tasks. Then $\Delta\mathbf{q}_1 = \mathbf{J}_1^\star\mathbf{e}_1$, where $\star$ is an inverse operator (commonly, the pseudoinverse). For each i from 2 to the number of tasks k, one can compute

$$\Delta\mathbf{q}_i = \Delta\mathbf{q}_{i-1} + (\mathbf{J}_i \cdot P_{i-1}^A)^\star(\mathbf{e}_i - \mathbf{J}_i\Delta\mathbf{q}_{i-1}), \tag{3.18}$$

where $P_{i-1}^A = I - \mathbf{J}_i^{A\star}\mathbf{J}_i^A$ is the kernel projection operator and $\mathbf{J}_i^A$ is an augmented Jacobian of the first i tasks

$\mathbf{J}_i^A = \left[\mathbf{J}_1; \quad \dots \quad ; \mathbf{J}_i \right]$.

For online position-tracking problems, a variant of this method can be implemented as explained in [94] to avoid singularity issues.

Task augmentation (TA)

Instead of projecting a secondary task onto the kernel of the robot position tracking task, in the case of a redundant robot a secondary task can be added as a new row of the Jacobian matrix to complete it and obtain a square matrix [95].

Completing the Jacobian matrix to a square matrix has the advantage of allowing to use its inverse directly when it is full rank. Nevertheless, it is possible that the added rows are linearly dependent on the geometric Jacobian rows. To overcome this problem, a general weighted Jacobian can be used [96] in which a *Gram-Schmidt* orthonormalisation is performed to ensure that the added task does not originate rank problems.

Moreover, an activation threshold may be used for the added task (e.g., when it entails avoiding joint limits).

3.1.4 Joint limit avoidance

Joints usually have limits on their prismatic/rotational ranges, and a solution to the IK with a joint value outside its limits is not a feasible solution. Hence one of the most important properties of a good IK solution is that it lies inside these limits. The redundant degrees of freedom of a robot are often used to achieve such a goal.

Joint limits as a secondary task

A first approach to avoid joint limits may be to use gradient projection. One can use a joint-centering function and project it to the kernel of the main task. For example, taking [97]

$$H = -\frac{1}{2m}\sum_{i=1}^{m}\frac{(q_i^{max} - q_i^{min})^2}{(q_i^{max} - q_i)(q_i - q_i^{min})}, \tag{3.19}$$

which tends to infinity when approaching a joint limit, and has a minimum value at the joint range's center.

Using the GP equation (3.17) with the gradient of the H function defined, joints are pushed to their range center values, but joint limits are not always avoided. This is due to the fact that the push-to-center function H is only activated on the kernel of the position reaching task. In [98], the obtained joint variations are rescaled at each iteration to keep joints within the valid range.

Avoiding joint limits with activation matrices

To avoid joint limits, one can also use a weighting matrix that penalizes the motion of the joints approaching a limit [92], or even block them. This is called Joint Clamping (JC).

In [99], a factor is added when updating the joint state at step k: $\mathbf{q}^{k+1} = \mathbf{q}^k + H\Delta\mathbf{q}^k$ is used instead of $\mathbf{q}^{k+1} = \mathbf{q}^k + \Delta\mathbf{q}^k$, where H is a diagonal activation matrix, with:

$$h_i = \begin{cases} 0 & \text{if } q_i > q_i^{max} \text{ or } q_i < q_i^{min} \\ 1 & \text{otherwise} \end{cases}$$

This control law clamps any motion that violates joints' limits, but does not push the joints away from them. In this way, when the robot reaches a joint limit, it may loose this degree of freedom, and go on with the others to reach the target. The algorithm follows the expression:

$$\Delta\mathbf{q} = H(\mathbf{J}H)^{\dagger}\mathbf{e}. \tag{3.20}$$

A problem that may arise when using this algorithm is that, even with an activation matrix continuous wrt. joint activation as in [99], the pseudoinverse operator is not continuous with respect to this activation matrix. Theorem 4.2 in [100] states that the effect of any nonzero h_i in (3.20) is the same (in the sense that there is no dependency on the value h_i as long as it is not zero). In fact, it can also be seen that damping the pseudoinverse does not solve the problem, outside of a very small interval [101]. To resolve these issues, this previous work proposes a continuous (wrt. activation matrix) pseudoinverse operator, defined as follows.

For task-activation matrices $G = diag(g_1, .., g_n)$:

$$\mathbf{J}^{\oplus G} = \sum_{P \in \wp(N)} \left(\prod_{i \in P} g_i \right) \left(\prod_{i \notin P} (1 - g_i) \right) \mathbf{J}_P^{\dagger}, \tag{3.21}$$

$\wp(N)$ being the power set of $N = \{1, .., n\}$, and $\mathbf{J}_P = G_0 J$, where G_0 is a square diagonal matrix with $G_{0_i} = 1$ if $i \in P$ and 0 otherwise.

And for joint-activation matrices $H = diag(h_1, .., h_m)$:

$$\mathbf{J}^{H\oplus} = \sum_{Q \in \wp(M)} \left(\prod_{i \in Q} h_i \right) \left(\prod_{i \notin Q} (1 - h_i) \right) \mathbf{J}_Q^{\dagger}, \tag{3.22}$$

$\wp(M)$ being now the power set of $M = \{1, .., m\}$, $\mathbf{J}_Q = \mathbf{J}H_0$, where $H_{0_i} = 1$ if $i \in P$, and 0 otherwise. The idea behind these pseudoinverse expressions is that the transition between activated and deactivated tasks is smooth wrt. the activation parameters h_i, g_i. For further development of these inverses, see [101].

Therefore, by using:

$$\Delta \mathbf{q} = \mathbf{J}^{H\oplus} \mathbf{e}, \tag{3.23}$$

the continuity problem is solved. However, when blocking a joint, the robot would still lose one degree of freedom and eventually may not reach the goal.

Joint limits as the primary task

Although one can add secondary tasks to avoid joint limits, the only way to guarantee such avoidance is to set it as the primary task and include the positioning goal as a secondary task. To this purpose, it is defined $\mathbf{J}_1 = H_m$, m being the number of joints of the manipulator, and matrix H_m as

$$H_m = diag(h_\beta(q_i)), \tag{3.24}$$

where h_β is a continuous function (usually a piecewise function, as in [86]) that progressively deactivates the joint when it reaches a specified distance β from its limits.

Then the main task error in (3.18) is defined as $\mathbf{e}_1 = -\lambda_{JL}\mathbf{q}$, λ_{JL} being a scalar to weigh the importance of joint limits. Typically, $\lambda_{JL} \in [0.1, 0.5]$.

With these definitions of $\mathbf{e}_1$ and $\mathbf{J}_1$, $\Delta\mathbf{q}_1$ is zero when the joints' positions are far from their limits (at a distance greater than β for each joint), and has a *push-to-center* value when in the limits neighborhood. On its kernel (i.e., the joints which are not forced to move to the center

due to their proximity to the limit), a secondary task is applied to reach the goal with $\mathbf{J}_2 = \mathbf{J}$, the Jacobian matrix of the robot, and $\mathbf{e}_2 = \mathbf{x}_d - \mathbf{x}$, the positioning error.

Then the algorithm following this task-priority hierarchy is:

$$\Delta\mathbf{q} = \mathbf{J}_1\mathbf{e}_1 + \mathbf{J}_2(I_m - \mathbf{J}_1^{\dagger}J_1)^{\dagger}(\mathbf{e}_2 - \mathbf{J}_2\mathbf{J}_1\mathbf{e}_1). \tag{3.25}$$

Moreover, this task-priority scheme can be performed with the continuous pseudoinverse operator [86], to get: $\Delta\mathbf{q}_1 = \mathbf{J}_1^{\oplus H}\mathbf{e}_1$, H being the same matrix defined in (3.24). And the following tasks would be defined as:

$$\Delta\mathbf{q}_i = \Delta\mathbf{q}_{i-1} + \mathbf{J}_i^{P_{\oplus}^{i-1}\oplus}(\mathbf{e}_i - \mathbf{J}_i\Delta\mathbf{q}_{i-1}), \tag{3.26}$$

where $P_{\oplus}^0 = I$, and $P_{\oplus}^i = P_{\oplus}^{i-1} - \mathbf{J}^{P_{\oplus}^{i-1}\oplus}\mathbf{J}_i$ is the *kernel projection* operator. For the case considered, (3.25) is equivalent to:

$$\Delta\mathbf{q} = \Delta\mathbf{q}_1 + \mathbf{J}_2^{P_{\oplus}^1\oplus}(\mathbf{e}_2 - \mathbf{J}_2\Delta\mathbf{q}_1), \tag{3.27}$$

with $\Delta\mathbf{q}_1 = \mathbf{J}_1^{H\oplus}\mathbf{e}_1 = H(-\lambda_{jl}\mathbf{q})$, as $\mathbf{J}_1 = I_m$ and $\mathbf{e}_1 = -\lambda_{jl}\mathbf{q}$. Therefore:

$$\Delta\mathbf{q} = H(-\lambda_{jl}\mathbf{q}) + \mathbf{J}_2^{(I_m-H)\oplus}(\mathbf{e}_2 + \lambda_{jl}\mathbf{J}_2H\mathbf{q}). \tag{3.28}$$

3.1.5 Second enhancement: pseudoinverse smoothing

The TP scheme may present large steps and gains, resulting in an almost-chaotic behavior. To solve these uncontrolled gains, it would be necessary to avoid large steps and condition numbers. Paying attention to (3.25), we can reorder the terms and separate the position error-dependent terms ($\mathbf{e}$) from those that don't depend on it:

$$\Delta\mathbf{q} = \left(I - \mathbf{J}^{(I_m-H)\oplus}\mathbf{J}\right)H(-\lambda_{jl}\mathbf{q}) + \mathbf{J}^{(I_m-H)\oplus}\mathbf{e}. \tag{3.29}$$

We intend to apply the ideas underlying SD [55], so as to damp selectively each one of the task space eigenvectors of the Jacobian matrix J, or its filtered version with SVF, taking care of the dependency of the position variation $J\Delta\mathbf{q}$ with respect to the position error $\mathbf{e}$.

To do so, we have to find a bound for $\mathbf{J}\Delta\mathbf{q}$, i.e., the position variation after each step, which can be written using (3.29) and separating the position error-depending part ($\mathbf{e}$) from the rest as follows:

$$\mathbf{J}\Delta\mathbf{q} = \mathbf{J}\left(I - \mathbf{J}^{(I-H)\oplus}\mathbf{J}\right)H(-\lambda\mathbf{q}) + \mathbf{J}\mathbf{J}^{(I-H)\oplus}\mathbf{e}. \tag{3.30}$$

Now, after calculating $\mathbf{J}^{(I-H)\oplus}$, we can use its SVD, keeping in mind that the result of this

decomposition has to be expressed knowing $\mathbf{J}^{(I-H)\oplus}$ is an inverse of $\mathbf{J}$, thus

$$\mathbf{J}^{(I-H)\oplus} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^T = \sum_{i=1}^{d} \hat{\sigma}^{-1}\mathbf{v}_i\mathbf{u}_i^T. \tag{3.31}$$

And knowing that $(\mathbf{u}_k^T \cdot \mathbf{e}) = (\mathbf{u}_k^T \cdot \sum_{s=1}^{d}(\mathbf{u}_s^T \cdot e)\mathbf{u}_s) = \sum_{s=1}^{d}(\mathbf{u}_s^T \cdot \mathbf{u}_k)(\mathbf{u}_s^T \cdot e)$ in the expression

$$\mathbf{J}^{(I-H)\oplus}\mathbf{e} = \sum_{i=1}^{r} \hat{\sigma}_i^{-1}\mathbf{v}_i\mathbf{u}_i^T\mathbf{e}, \tag{3.32}$$

we can take, by analogy to the SD algorithm, for $\mathbf{e} = \mathbf{u}_s$, the joints variation $\Delta\mathbf{q}^s$ used by SD as:

$$\mathbf{J}^{(I-H)\oplus}\mathbf{u}_s = \hat{\sigma}_s^{-1}\mathbf{v}_s \Rightarrow \Delta\mathbf{q}^s = \hat{\sigma}_s^{-1}J\mathbf{v}_s \tag{3.33}$$

which has an effect on the jth joint of:

$$\Delta q_j^s = \hat{\sigma}_s^{-1}\mathbf{J}^j v_{j,s}, \tag{3.34}$$

where $v_{j,s}$ is the jth position on the sth column of matrix V, and $\mathbf{J}^j$ is the jth column of matrix $\mathbf{J}$.

Therefore, adding the norms for all joints we get the bound M_s as defined in [55]:

$$\sum_{j=1}^{m} |\Delta q_j^s| \leq \hat{\sigma}_s^{-1} \sum_{j=1}^{m} |v_{j,s}| \|\mathbf{J}^j\| = M_s. \tag{3.35}$$

This M_s is a bound on the position change gain in the task space generated by the error-dependent part of the algorithm, for each component of the error, and thus with it we can set, for each $s = 1..n$, the maximum joints change γ_{max}:

$$\gamma_s = min(1, 1/M_s)\gamma_{max}, \tag{3.36}$$

to then proceed exactly as in SD:

We will first compute the joints change for each error component (m-dimensional vector):

$$w_s = \hat{\sigma}_s^{-1}\mathbf{v}_s \left(\mathbf{u}_s^T \cdot \mathbf{e}\right), \tag{3.37}$$

and we will bound this variation with the γ_s obtained in (3.36):

$$\Delta q_s = \begin{cases} 1 & \text{if } \|w_s\| < \gamma_s \\ \frac{ws}{\|ws\|}\gamma_s & \text{if } \|w_s\| \geq \gamma_s \end{cases} \tag{3.38}$$

Now, differing from SD algorithm, we have to add the non error-dependent part of the algorithm to the sum of each component :

$$\Delta\hat{\mathbf{q}} = (I - \mathbf{J}^{(I-H)\oplus}\mathbf{J})H(-\lambda\mathbf{q}) + \sum_s \Delta q_s, \tag{3.39}$$

to finally bound the total joint variation by γ_{max}:

$$\Delta\mathbf{q} = \begin{cases} 1 & \text{if } \|\Delta\hat{\mathbf{q}}\| < \gamma_{max} \\ \frac{\Delta\hat{\mathbf{q}}}{\|\Delta\hat{\mathbf{q}}\|}\gamma_{max} & \text{if } \|\Delta\hat{\mathbf{q}}\| \geq \gamma_{max} \end{cases} \tag{3.40}$$

In this way, we ensure that $\Delta\mathbf{q}$ is bounded, respects joint limits, and it is sufficiently well-conditioned.

The described joint limit-concerned methods have also been compared on an example trajectory in the 4R manipulator in order to check which ones respect joint limits and which do not, as displayed in Fig. 3.4. In all cases the joint limits are sometimes surpassed, as the push-to-center value action is done *a posteriori*, but the following iteration pushes this joint value back to its feasible interval. As expected, neither JW, nor GP or TA are capable of successfully avoiding joint limits. On the other hand, JC is uneffective because it tends to block joints, losing degrees of freedom.

Furthermore, CTP, which continuously activates/deactivates joints, is not capable of converging nor of maintaining a smooth trajectory. Our proposal solves these problems and significantly improves performance.

3.1.6 Experimentation

As a benchmark to test both the reviewed and proposed algorithms, all of them have been implemented in *Matlab* and C++ (using a ROS library) in a 7-DoF redundant WAM robot arm (with Denavit-Hartenberg parameters as shown in Table 3.3) and their performance has been tested as global IK solvers. To do so, 1000 random feasible initial and target positions have been generated, using a uniform probability distribution for all joints within their limits, and mapped into a Cartesian position with the forward kinematics function.

Note that, for the algorithms taking into account joint limits, this sampling as such was not adequate to assess their performance, as the arm has several assembly modes (different arm

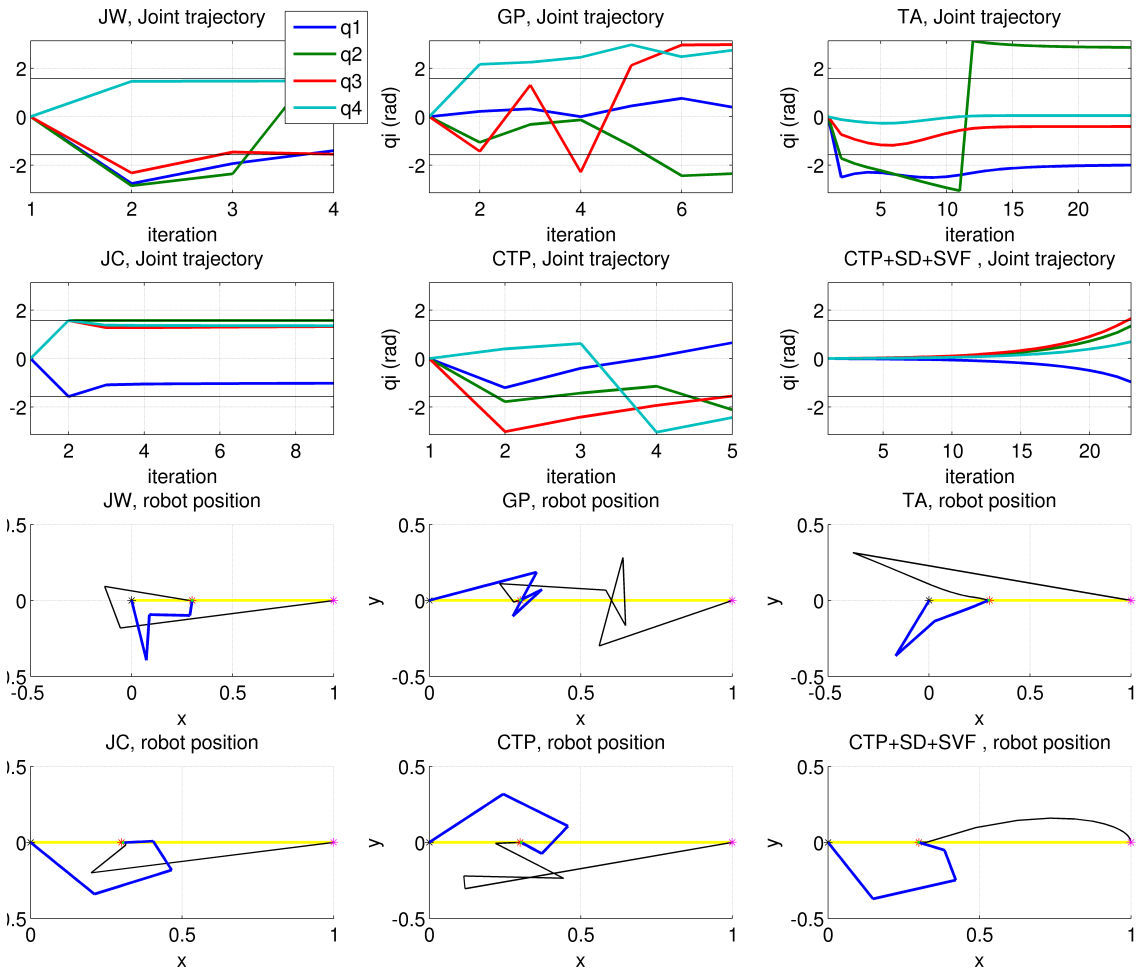


Figure 3.4: Behavior of some CLIK algorithms (see Table 7.1 for notation) applied to a 4R planar manipulator. The robot frame evolves from a light color start position (horizontal) to a darker ending one, with its end-effector's trajectory plotted in black (thin line). Note the different scales for the x-axis. In the CTP algorithms, the joints have been allowed to cross joint limits to show how they are capable of moving back to a centered joint value.

Table 3.3: Denavitt-Hartenberg Standard parameters for the WAM robot arm, where $d_3 = 0.55$, $d_5 = 0.3$ and $d_7 = 0.06$.

link	a_i	α_i	d_i	q_i	q_i^{min}	q_i^{max}
1	0	$-\pi/2$	0	q_1	-2.6	2.6
2	0	$\pi/2$	0	q_2	-2.0	2.0
3	a	$-\pi/2$	d_3	q_3	-2.8	2.8
4	-a	$\pi/2$	0	q_4	-0.9	3.1
5	0	$-\pi/2$	d_5	q_5	-4.8	1.3
6	0	$\pi/2$	0	q_6	-1.6	1.6
7	0	0	d_7	q_7	-2.2	2.2

configurations for the same end-effector position), which added to the joint limits restriction, could make the desired configuration sometimes impossible to reach with the same assembly mode as the initial configuration. This fact has an impact on CLIK algorithms by sometimes requiring very different initial and final joint configurations, leading to moving the arm in unintiutive ways so as to have feasible solutions. In these cases, a path planner is needed, which is out of the scope of this study.

To avoid the mentioned situations, in the experiments involving algorithms taking into account joint limits (those in Table 3.5), we generated the samples by obtaining an initial joint position $\mathbf{q}_0$ and a final one $\mathbf{q}_F$. If $|\mathbf{q}_0^i - \mathbf{q}_F^i| < A$, $\forall i = 1..7$, for a given constant value A, the sample was accepted. This ensures that there exists a solution of the desired position in the same assembly mode or another assembly mode close to it. This A parameter will be used later to check the sensitivity of the different algorithms to the initial/ending position distance, as shown in Fig. 3.5.

The results of a *Matlab* simulation can be seen in Tables 3.4 and 3.5, where the columns in Table 3.4 show the percentage of solutions found, the average computation time ($\bar{t}_{sol}$), the remaining error when convergence was not achieved ($\bar{e}_{nosol}$), and the average number of iterations needed to find a solution ($\overline{it}_{sol}$). In Table 3.5 an additional column (second) shows the percentages of solutions where joint limits were respected. As the algorithms in Table 3.4 do not consider joint limits, we only show this information in the second Table. The performance of the reviewed state-of-the-art methods is compared with our proposals, which are highlighted in bold face in the Tables. Besides the filtering enhancement SVF in different combinations, we have used the CTP algorithm in (3.28) together with the SD, as proposed in Section 3.1.5, and we have also combined them with SVF to compare results. With these data, we can draw the following conclusions:

Table 3.4: Behavior of the studied methods not concerned with joint limits for a sample of 1000 random initial and end positions for the WAM robot arm. Notation as in Table 7.1.

Method	% sol.	$\overline{t_{sol}}$ (ms)	$\overline{e_{nosol}}$	$\overline{it_{sol}}$
JP	100.0	42.6	-	12.2
JT	40.70	504.8	0.302	148.7
SD - $\gamma_{max} = 0.5$	98.4	154.3	0.042	43.5
JD - $\lambda = 0.005$	100.0	39.4	-	11.6
JF - $\lambda_{max} = 4\lambda$	100.0	38.6	-	11.2
ED	100.0	36.9	-	10.6
IED - $\Omega = 0.01 I_m$	100.0	38.7	-	11.1
SVF - $nu = 10$, $\sigma_0 = 0.01$	**100.0**	**37.1**	**-**	**10.7**
SVF+ED	**100.0**	**35.8**	**-**	**10.3**
SVF+SD	**99.7**	**145.2**	**0.041**	**41.1**

Table 3.5: Behavior of the studied methods taking into account joint limits for a sample of 1000 random initial and end positions for the WAM robot arm. Notation as in Table 7.1 and parameter A=1.0 rad.

Method	% sol.	% sol(JL)	$\bar{t}_{sol}$ (ms)	$\overline{e}_{nosol}$	$\overline{it}_{sol}$
JW - as in [92]	100.0	45.3	37.5	-	9.6
GP - $\mu = 0.2$	100.0	34.7	43.0	-	11.0
TA - as in [96]	100.0	84.6	147.3	-	28.3
JC - H as in [86]	73.6	53.5	73.1	0.826	18.9
TP - H as in [86]	33.6	33.6	48.9	1.005	12.0
CTP - H as in [86]	83.7	83.7	366.8	0.381	21.8
CTP+SVF	**86.6**	**86.6**	**300.4**	**0.401**	**48.2**
CTP+SD	**97.1**	**97.1**	**660.0**	**0.867**	**26.2**
CTP+SD+SVF	**98.3**	**98.3**	**568.7**	**0.719**	**22.6**

- Low convergence ratio of JT. This is due to chattering when activating/deactivating joints, as commented before. The remaining algorithms not considering joint limits always converge, except for SD, due to the limited number of iterations.

- Using SVF improves the speed of JP, requiring less iterations on average and, combined with ED, performs much faster than the rest of methods. Nevertheless, we also recommend using SD+SVF, because this guarantees the steps will always be smooth, even in the case of a singular goal position.

- JW, TA and GP do not respect joint limits. This is due to the fact that avoiding limits is not treated as a priority, thus zero-error positioning prevails.

- All the algorithms not fully respecting joint limits have higher convergence ratios, since they can cross regions with unfeasible joint values to reach the goal.

- The TP algorithm does not converge most of the times. This is due to the discontinuity commented before, causing large gains which then block the joints.

- CTP algorithms do not always converge, but when they do, the solution respects joint limits. This shows that using these limits as a primary task is a successful strategy. Adding SD improves the convergence ratio, and it also reduces computation time. Overall, CTP computation times are large, due in part to the non optimality of *Matlab* for its calculation. This could be reduced by finding an approximate value of the continuous pseudoinverse.

The non-convergence cases of our CTP proposed algorithms are due to algorithmic singularities. These happen when, close to a joint limit, the push-to-center value of the joint limit avoidance task compensates the position tracking error. This is like saying that the algorithm walks into a dead end in joint space. To avoid this convergence problem, some works in literature try to find a better initial point through a biased random sampling over other possible starting configurations, or it is also possible to use a path planning algorithm in order not to get stuck.

To further test our proposals, we repeated the experiment in Table 3.5 with different values of the parameter A defined before. Figure 3.5 shows the percentage of solutions found within joint limits for each algorithm and each value of A. There we can see that SVF improves the performance of the CTP algorithms, and that the combination of CTP, SD and SVF outperforms the other algorithms in literature. In addition, we see that some algorithms without the selective damping have worse results for very small values of A; this is because they have large gains, as explained along this work, and they sometimes surpass the goal.

3.1.7 Discussion

In this section, the most relevant CLIK algorithms for redundant robots have been compared. Special attention has been paid to three issues: The JP algorithm may have very large gains along certain directions, and reducing the global gain is not the best solution. In fact, having such large gains is similar to a random positioning in joint space, whose topology is equivalent to an *m-torus* mapped into the workspace, and the high convergence rate of JP in Table 3.4 is due to the fact that large steps are taken until the end-effector reaches a position from which the goal is achievable. The JT algorithm does not have such problem, but in some cases presents so much

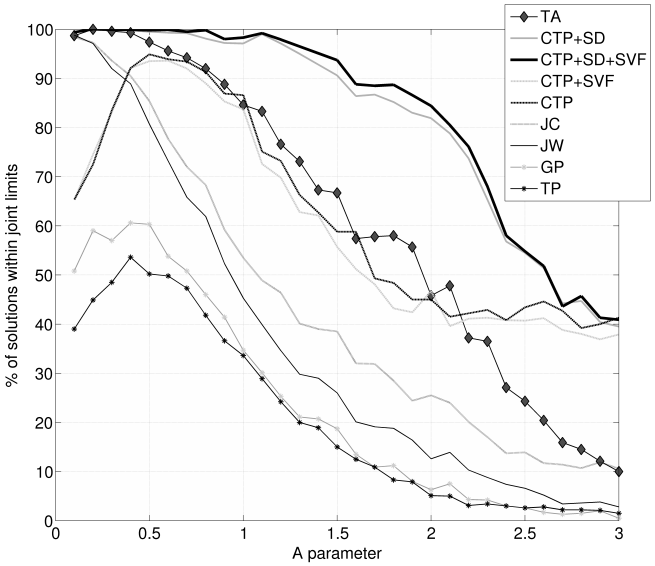


Figure 3.5: Percentage of solutions found within joint limits for different sampling thresholds A, an indicator of the distance between initial joint position and goal joint positions.

chattering that makes its computational cost grow. Since SD efficiently solves this problem, it is recommended to use such damping in most algorithms.

Looking for good matrix conditioning, we have compared the capability of the different algorithms to avoid amplifying the numerical error in robot positioning. The outcome has been that most of the existing methods do not perform well near a singularity. Filtering or damping the Jacobian matrix improves this conditioning, but with no numerical guarantees. On the other hand, using the current error as a damping factor reduces the condition number, but when close to the goal, the ED algorithm (or its improved version, IED) behaves similarly to filtering or damping.

Therefore, we proposed a new filtering method based on a continuous modification of the singular values of the Jacobian, which we named SVF. We proved theoretically and in practice that our proposal improves the existing methods for numerically filtering or damping the Jacobian pseudoinverse of a matrix. We have also shown that this does not entail a significant growth in the computational cost. With this filtering, the Jacobian matrix can be assumed to be always full rank, without generating much additional error on the algorithms, thus the pseudoinverse operator would not have discontinuities due to a rank change in the Jacobian matrix. This can be used in all control-based methods to improve their performance.

We have also presented a review of first-order approaches to achieve secondary tasks. In particular, we have tried to devise an algorithm that efficiently avoids joint limits. Through

experimentation, we have seen that the only way to ensure avoiding such limits is to treat them as the main priority task by adding an activation matrix on this main task. This then results in discontinuities of the pseudoinverse operator when activating or deactivating a joint push-to-center value to avoid a joint limit. However, this shortcoming is solved with the continuous pseudoinverse (CTP) which, when combined with SD and our proposed filtering (SVF), ensures controlled steps and a full-rank behavior of the Jacobian.

As it is well-known, and it showed up in our testing with a redundant robot such as the Barrett WAM arm, CLIK methods used as global IK solvers do not always reach the goal. This is because of algorithmic singularities, i.e., when the main task and the secondary task compensate one another and the computed joint variation becomes zero. To solve this issue, it is recommended to add a path planner to the algorithm or a randomized initial value to iterate upon, so as to prevent the robot getting stuck in such a situation. Despite the convenience of a path planner to obtain smoother joint changes, our last proposal (CTP+SD+SVF) performs well without such a planner, keeping over a 90 percent success rate with parameter $A \leq 1.5rad$ in Fig. 3.5, it being the best among all the tested algorithms. Additional experimentation using such algorithms in a trajectory-tracking experiment can be found in Appendix B.1.

Having analysed the kinematics capabilities of a robot, a continuation of such work was to determine how well two robots could interact with each other.

3.2 Bimanual arm positioning

Bimanual manipulation of objects is receiving a lot of attention nowadays, but there is few literature addressing the design of the arms configuration.

We investigated a way of deciding the arms relative position, depending on the task, by fully characterizing manipulability in the workspace of the Barrett WAM arm. Using the fact that the robot has a spherical wrist, we propose to compute its feasible orientations for each Cartesian point and pack them in a bounding cone to obtain an easy characterization of robot feasible poses. After having characterized the workspace for one robot arm, we can evaluate how good each of the discretized poses relate with an identical arm in another position with a quality function that considers orientations. In the end, we obtain a quality value for each relative position of two arms, and we perform an optimization using genetic algorithms to obtain the best workspace for a cooperative task.

In Section 3.2.1 we will explain how we characterize the workspace with information on all feasible orientations and how we store these data. This is later used in Section 3.2.2 to evaluate such relative positioning of two identical arms. Finally, in Section 3.2.4 we describe the implementation, and we use a genetic algorithm to search for the best relative positioning

of the two arms.

3.2.1 Workspace representation

For a redundant robot, it is well known that the Forward Kinematics (FK) function f is not one-to-one, and given its non-linearity, the workspace may be hard to represent. The Inverse Kinematics (IK), the inverse of the FK, may then be better to characterize the workspace. Note that, for each point in the Cartesian space, more than one IK solution may exist [14]. We initially decided to characterize the workspace numerically, as a subset of $\mathbb{R}^3 \times SO(3)$, by discretizing it. To do so, a uniform mesh is set for the Cartesian position and/or orientation, and, for each point P_i on the mesh, the existence of a joint solution q^{P_i} such that $P_i = f(q^{P_i})$ is checked. This can be done by sampling the joint space and using the FK function (forward sampling), or by sampling the workspace and using the IK (inverse sampling). Nevertheless, while the forward sampling results in a biased sampling of the workspace, the inverse is able to exhaustively analyze the whole workspace, thus we recommend this option if a good IK algorithm is available (for the case of the WAM robot, the IK can be obtained either by iterative methods [14] or analytical methods [28, 29]).

To plot the reachable positions of the workspace, and store its data, we used a similar method as in [73]. For each point of the $3D$ mesh representing the workspace, M solutions of the IK of the robot, with different orientations, are obtained. To ensure a good distribution of these orientations, we can use the proposal in [102], where points are arranged in hexagonal patterns to fit on the sphere, or use randomly generated quaternions. If there exists at least one solution, the position is reachable. In addition, for each of these M IK attempts, we can extract additional information, such as manipulability [57] at the obtained pose, percentage of orientations found for a given $3D$ Cartesian point, etc.

For the feasible orientations of a robot arm in a Cartesian point, several geometrical shapes to represent the valid orientations have been proposed in literature [73]. Among these shapes, cones are probably the best choice, due to their simplicity and easy characterization. In fact, for a robot with a spherical wrist (see Fig. 3.6a), the Tool Center Point (TCP) stays within a cone whose axis is the rotation axis of the first degree of freedom of the wrist (namely, the forearm axis).

Moreover, discarding the rotation around the TCP z-axis, we propose to collect the set of valid forearm axes at a certain Cartesian point $P \in \mathbb{R}^3$, which will be enclosed by a cone, and compute the Bounding Cone (BC) that contains them all with the algorithm proposed in [103]. Also, if the wrist angle has symmetric limits, its aperture can be added to the BC angle, yielding a cone that contains all the TCP z-orientation axis that the robot can reach at the given position (see Fig. 3.6b).

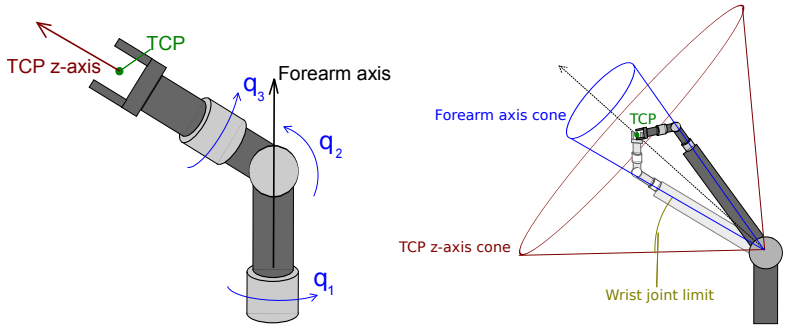


Figure 3.6: Left: Spherical wrist. q_2 is the wrist angle. Right: Robot scheme showing the bounding cone of all possible forearm axes. This cone is augmented by adding the wrist joint limit to obtain the possible TCP z-axis bounding cone.

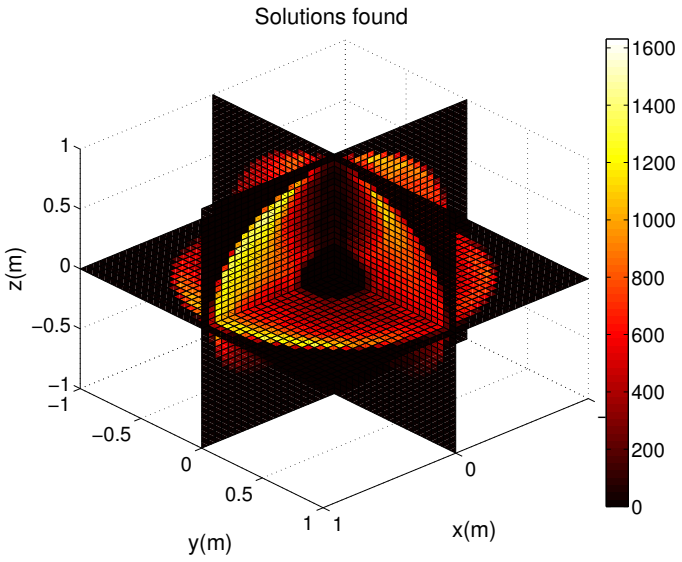


Figure 3.7: Solutions found over the workspace of a WAM robot.

With this approach, we obtain a mesh for the workspace, encoding all the information gathered when computing the reachable positions such as manipulability, percentage of solutions found, etc. plus the obtained bounding cone containing all the possible z-axis of the TCP. We can see an IK solutions map over the workspace of a WAM robot (see Table 1 in [29] for its dimensional parameters) in Fig. 3.7.

3.2.2 Bimanual workspace

Multiple-arm cooperative tasks provide the capability of performing tasks that would be impossible or, at least, much more difficult to accomplish with only one arm. Although actuating the arms to simultaneously move an object may be a hard task, the arms relative configuration must be given importance, since depending on the intended use of bimanual robots, some configurations might be better suited than others. Human arms configuration may be the best for the tasks performed by humans along their evolution, with a large workspace in front, and a very reduced workspace at our back, as our attention and visible space stays in front of us.

In [68], a review of bimanual manipulation is done, where the state-of-the-art in cooperative tasks is analyzed. Some examples of existing bimanual robots are shown, such as the *Justin robot* [72], where the arms are placed in a humanoid-like configuration with a tilt of 60 degrees, the DARPA arm robot [104], with two Barrett WAM robot arms with their bases placed perpendicular, or the ARMAR [105] [106] robot, where both z base axes are placed in an aligned humanoid-like configuration.

However, the humanoid configuration may not be the best one for certain tasks. In this section, we provide hints to determine how good is a relative positioning of robot arms, in the form of a parametrized value function to be used with a numerically characterized workspace as that in the previous section.

3.2.3 Proposed quality function

In this chapter, we intend to characterize a common workspace between two arms. To this purpose, given two Cartesian points $P_1 \in W_1$, $P_2 \in W_2$, we will compute several factors that will lead to a quality function for each pair (P_1, P_2) defined as:.

$$F(P_1, P_2) = DF \cdot SDF \cdot OF \cdot MF \cdot CF,$$

where DF is the Distance Factor, SDF the Solutions Density Factor, OF the Orientation Factor, MF the Manipulability Factor and CF the Conditioning Factor. Multiplication and not addition of factors has been chosen to strongly penalize those positions with very low value on one factor.

Then, the global quality value of a relative position of two arms is:

$$F = V \cdot \sum_{P_1 \in W_1} \sum_{P_2 \in W_2} F(P_1, P_2), \tag{3.41}$$

V being the total volume of the combined workspace. Evaluating this quality measure (3.41) we get a mapping $g : \mathbb{R}^3 \times SO(3) \rightarrow \mathbb{R}$, which maps a relative position plus orientation transformation (up to 6 variables) to a real value. This mapping can be used by genetic algorithms to search for its maximum, which would correspond to the best relative positioning.

Points to compare and distance factor

For each pair of points (P_1, P_2), we have to decide whether to evaluate their relation or not. In order to decide that, we may take a characteristic length L for the object to be manipulated, and then one possible way to evaluate that relation is by using the following Distance Factor (DF):

$$DF = \begin{cases} 1 & \text{if } L - \delta < \|P_2 - P_1\| < L + \delta, |a_1| < \alpha_1, \text{ and } |a_2| < \alpha_2 \\ 0 & \text{otherwise} \end{cases}, \tag{3.42}$$

where δ is a tolerance on the manipulated object length, and $a_1, a_2, \alpha_1, \alpha_2$ are defined with the bounding cones in Fig. 3.8. If the segment joining P_1 and P_2 does not lie within both orientation cones for P_1 and P_2, their relation may not be evaluated. However, the orientation restriction can be made more permissive, depending on the kind of graspings to perform.

Solution density factor

As defined in some previous works [70, 73], the SDF is the ratio of the IK solutions found over the attempted solutions. For each Cartesian point of the workspace, we retain the percentage of IK solutions found, given random orientations. The SDF is then defined as the product of the ratios for the two points compared.

Orientation factor

Imagine two arms manipulating an object of length L grasped at points $P_1 \in W_1$ and $P_2 \in W_2$. In this situation, a grasp in which the TCP z-axis of each arm is aligned in the direction of the other grasping point is usually preferred. This can be checked using the cones (Z_1, α_1), (Z_2, α_2) obtained for each workspace: we can calculate the angles a_1, a_2 from the vector $P_1 - P_2$ and the cones axes Z_1, Z_2, as in Fig. 3.8.

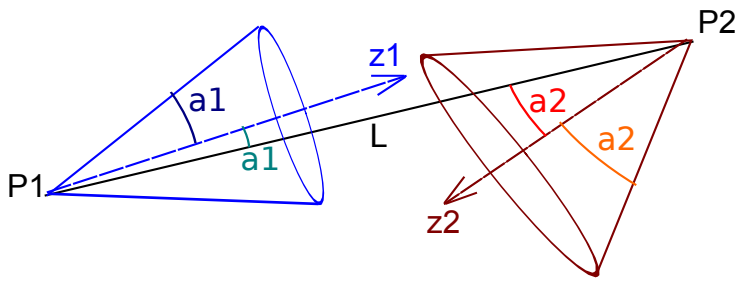


Figure 3.8: Distance and Orientation factor variables: α_1, α_2 are the cones angles, and a_1,a_2 the angles between each cone's axis and the line P_1P_2.

Then, we define the Orientation Factor (OF) for the points to compare as

$$OF = max\left(OF_{min}, 1 + \frac{1}{K} ln\left(\frac{\alpha_1 - |a_1|}{\alpha_1} \cdot \frac{\alpha_2 - |a_2|}{\alpha_2}\right)\right) \tag{3.43}$$

where K is a tuning parameter, and OF_{min} is the minimum value accepted for the orientation factor.

Thus defined, this factor verifies that $OF_{min} \leq OF \leq 1$, $\forall a_1, a_2, \alpha_1, \alpha_2$, when satisfying the conditions in (3.42), it having a value of 1 when both cones' axes are parallel to the vector $P_2 - P_1$ and pointing towards each other, and gradually reducing its value to OF_{min} when the axes point away from each other.

Manipulability factor

When performing cooperative tasks, or grasping an object with multiple arms, there are approaches to obtain a combined manipulability [107, 108]. However, combined manipulability computation for multiple arms holding an object relies on the arms poses, which are unknown for our workspace representation, as we use the average of many IK computations with different orientations and the redundancy of the robot gives us infinite solutions. So we take the average manipulability of both grasping points as a good approach to evaluate how manipulable is an object. In [70], grasping point candidates are selected based on this manipulability, so the MF is defined as:

$$MF(P_1, P_2) = \overline{m}(P_1)\overline{m}(P_2) \text{ , with } \overline{m}(P_i) = \frac{1}{M}\sum_{j=1}^{M} m(IK(P_i, o_j)), \tag{3.44}$$

$m(IK(P_i, o_j))$ being the manipulability at the joint position obtained as an IK solution for the robot at position P_i with orientation o_j.

Conditioning factor

In order to ensure a stable behavior for the related points, we use the Jacobian Condition Number (CN), which is defined as $\kappa(J) = \sigma_1/\sigma_d$, where σ_1, σ_d are the largest and the smallest singular values of the robot Jacobian matrix. The CN is a measure of the error amplification induced by the Jacobian matrix. Thus, we define

$$CF = \kappa(P_1) \cdot \kappa(P_2), \tag{3.45}$$

where $\kappa(P_i)$ is the average CN for the solved IK of P_i.

3.2.4 Experimentation

As a first application, we searched for optimal relative positioning of two WAM robots. To do so, we used genetic algorithms instead of performing an exhaustive analysis in order to obtain the results faster, using 10 generations of 20 elements each, and a probability of mutation on each variable slightly decreasing after each generation.

We considered valid objects for grasping those of a size between $0.3m$ and $0.5m$ for the DF and a $K = 2$ for the OF. We collected the best half at each generation, paired them, and created 20 new configurations for the next generation.

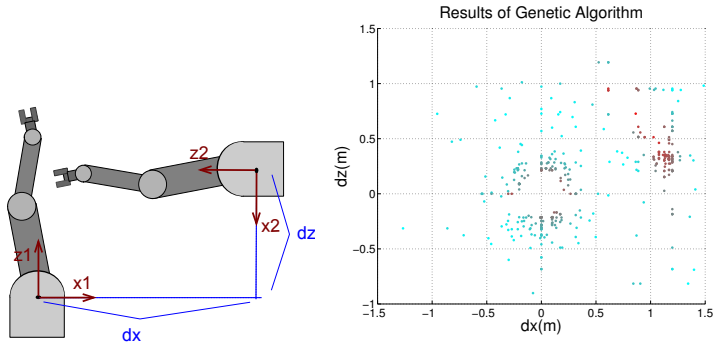


Figure 3.9: Left: First experimental settings. Right: Results with quality values from blue (low value) to red (high value).

Several settings were considered for optimizing a two-dimensional relative position between the arms. The first one is similar to that of the DARPA robot, with both z axes perpendicular. We found (see Fig. 3.9) that, for our criterion, the best configurations are those with positive dx and dz, while the DARPA robot has both negative offsets, which yield a lower value of the quality function. The best solution is for $dx = 0.8m$, $dz = 0.8m$ in which both arms cooperate at a larger distance than the DARPA robot. Other experimentation such as placing both arms with

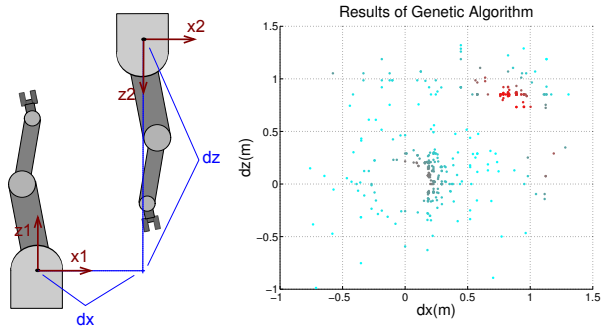


Figure 3.10: Left: Second experimental settings. Right: Results with quality values from blue (low value) to red (high value).

their z-axes parallel and facing each other (see Fig. 3.10) also leads to good positioning which might not have been considered when building bimanual robots.

3.3 Summary

Motivated by the need of a robust and practical Inverse Kinematics (IK) algorithm for the WAM robot arm, we reviewed the most used Closed-Loop IK (CLIK) methods for redundant robots, analyzing their main points of concern: convergence, numerical error, singularity handling, joint limit avoidance, and the capability of reaching secondary goals [15]. As a result of the experimental comparison, we have proposed two enhancements. The first is a new filter for the singular values of the Jacobian matrix that guarantees that its conditioning remains stable, while none of the filters found in literature is successful at doing so. The second is to combine a continuous task priority strategy with selective damping to generate smoother trajectories. Experimentation on the WAM robot arm has shown that these two enhancements yield an IK algorithm that improves on the reviewed state-of-the-art ones, in terms of the good compromise it achieves between time step length, Jacobian conditioning, multiple task performance, and computational time, thus constituting a very solid option in practice. This proposal is general and applicable to other redundant robots.

We have also presented a novel way to store the workspace information (including orientation, which is often not considered) of a robot with a spherical wrist, in a very compact way, thanks to the efficient bounding cones representation of the end-effector z-axis. This then allows us to evaluate the capability of a dual-arm robot to manipulate an object of a certain size, depending on the relative position of both arms. We can compute a global quality measure for a given relative position, in order to quantify how good a dual-arm configuration is. We used this quality measure to obtain better relative configurations with the help of a genetic algorithm.

And the results in Section 3.2.4 seem to indicate that, for the tasks studied, the configuration of current bimanual robots may not be optimal. The arm configurations of humanoid robots are designed to accomplish a wide repertoire of tasks while obeying diverse design and operational constraints. However, in settings with two independent robot arms, it may be simple and advantageous to tailor their relative configuration to the specificity of each particular task, as shown in the current work. The proposed algorithm can be further used to optimize relative positionings with more than two parameters in order to get more general results.

In the following chapter, we move on to integrate dynamics in this kinematic framework.

4

Robot Compliant Control

Robot compliant control aims at designing controllers that can accommodate deviations. Contrary to a *stiff* control, where a robot will track the desired position commands and try to compensate any deviation from such reference position, a compliant controller will allow deviations from such reference position. Such deviations are slightly compensated to try to reach the desired command, but usually with a significantly smaller gain that allows external agents to safely interact with robots without fear of accidents due to backlash movements, as well as limiting the strength with which objects are manipulated [109].

Compliant control allows robots to enter in human environments, as well as interacting with fragile and deformable objects. The idea behind compliant control is, throughout this thesis, to build an inverse dynamic model, i.e., to model the necessary torques satisfying the dynamics equation with position, velocity and acceleration commands, that will allow the robot to minimize the gains of the error-compensating control signal while tracking the desired commands. To do so, a prior knowledge of the dynamics of the robot is essential. However, many robots do not provide velocity/acceleration measurements, nor torque encoders in their joints. Therefore, those have to be obtained by position differentiation - resulting in noise amplification - or from the motor commands sent to the robot.

This chapter is divided in four sections. Section 4.1 presents a method to estimate external forces exerted on a manipulator during motion, avoiding the use of a sensor. This method is used together with a computed torque control scheme that compliantly follows a trajectory. A first iteration of the method is based on learning a task-oriented dynamics model and on a robust disturbance state observer. Such combination leads to an efficient torque observer that can be incorporated into any control scheme. However, locally learning the inverse dynamics of a robot results in the need of running motions before real executions, in order to learn the local dynamics of such motion.

To solve that issue, in Section 4.2, an analytical specific friction model for the Barrett's WAM

robot is built, showing a better behavior in the whole working space of the robot, i.e., a global friction model. Given that the other terms of the dynamics equation of the robot are generally known for the WAM robot case, building such friction model allowed us to perform compliant control while getting external forces feedback in the whole workspace.

Section 4.3 presents some applications where the presented framework is used either for tracking an object given visual information or to use policy search reinforcement learning to learn tasks involving deformable objects. Section 4.4 concludes the chapter with some final remarks.

4.1 External force estimation

In this section, we propose an approach based on machine learning techniques and disturbance state observers for the estimation of external forces/torques felt by the robot during common motion tasks. The presented method extends the state-of-the-art on external forces estimation to those cases where an analytical model of the robot dynamics is not available/feasible. Moreover, we took into consideration the well-known issue that the use of accelerations is undesirable due to the error introduced by the numerical differentiation, to elaborate a better contact force estimator, which will be incorporated in parallel to the controller as shown in Fig. 4.1.

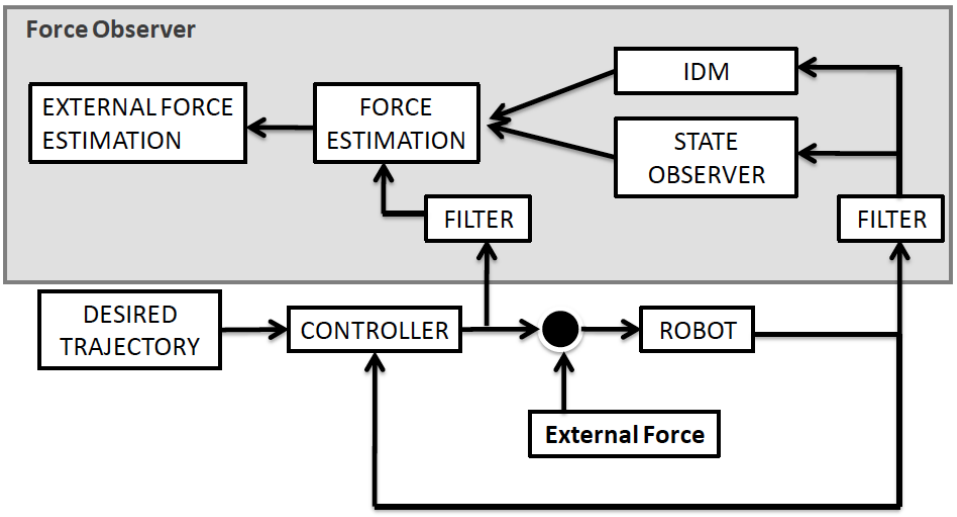


Figure 4.1: The proposed scheme can run in parallel to any controller.

4.1.1 External wrench estimation as a disturbance observer

This section describes how the proposed function approximation can be used to estimate the external wrench when present. Recalling the robot dynamics equation, Eq. (2.10):

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{F_f}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{u_T}, \tag{4.1}$$

it can be rewritten as

$$\begin{array}{rcl} \dot{\mathbf{x}}_1 & = & \mathbf{x}_2 \\ \dot{\mathbf{x}}_2 & = & \mathbf{\Gamma}(\mathbf{u}_c,\mathbf{x}) - \mathbf{M}^{-1}(\mathbf{x_1})\mathbf{u}_e \end{array}, \tag{4.2}$$

where $\mathbf{x} = [\mathbf{x}_1 \quad \mathbf{x}_2]^T$, with $\mathbf{x}_1 = \mathbf{q}$ and $\mathbf{x}_2 = \dot{\mathbf{q}}$. Here, accelerations due to external forces are separated from those produced by gravity, Coriolis, internal friction and torque commands, which gather in the term:

$$\mathbf{\Gamma}(\mathbf{u}_c,\mathbf{x}) = \mathbf{M}^{-1}(\mathbf{x}_1)\left[\mathbf{u}_c - \mathbf{n}(\mathbf{x_1},\mathbf{x_2})\right] \tag{4.3}$$

where $\mathbf{\Gamma}$ can be evaluated with the measurements of $\mathbf{u_c}$ and the learned function $\mathbf{n}$.

In [80] a force estimator is presented, which computes a state observer with gain $\mathbf{K}$ and deduces that the position error from the observer is due to an external force that can be computed as the *missing* force to make the observer perfectly track the state:

$$\mathbf{\Delta}_2\ddot{\mathbf{e}}_1 + \mathbf{\Delta}_1\dot{\mathbf{e}}_1 + \mathbf{\Delta}_0\mathbf{e}_1 = \mathbf{u}_e, \tag{4.4}$$

where $\mathbf{e}_1$ is the position estimation error, $\mathbf{\Delta}_2$ the inertia matrix, $\mathbf{\Delta}_1$ various Coriolis and friction terms, and $\mathbf{\Delta}_0 = -\mathbf{K}$, thus at stationary response, the external force in the robot's operational space is

$$\mathbf{f}_e = -\mathbf{J}^{T^\dagger}(\mathbf{x}_1)\mathbf{K}\mathbf{e}_1.$$

This observer does not assume a measurement of the joint velocities. However, it has the following drawbacks:

- It needs to compute the Coriolis matrix for different input values, and also the friction effects separately.
- It assumes a perfect model of the manipulator.
- No hints on the observer gains, $\mathbf{K}$, are given.
- Eq. (4.4) is not necessarily stable as defined. This equation is analyzed in [110], where the restrictions for its convergence are given. Nevertheless, its convergence radius depends

directly on the eigenvalues of an unknown matrix.

- A good value of the external force is only guaranteed at steady state, when $\dot{\mathbf{e}}_1 \simeq 0$ and $\ddot{\mathbf{e}}_1 \simeq 0$. Otherwise, as the term $\mathbf{\Delta}_1$ in equation (4.4) is very hard to learn or measure for learning purposes, the force estimation may have a large error. This results in a very slow response to external force steps, which can be seen in [80].

Proposed observer

Keeping in mind the issues in [80], we thought to treat the external force as a disturbance of the dynamic system, and use a disturbance observer. To this purpose, in [111] a state observer for dynamic systems is proposed that also estimates external unmodelled disturbances. To follow this we can rewrite equation (4.2) as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(\mathbf{x})\mathbf{d} + \mathbf{\Gamma}^*(\mathbf{u}_c, \mathbf{x}), \tag{4.5}$$

with $\mathbf{d} = -\mathbf{u}_e$, $\mathbf{A} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ \mathbf{M}^{-1}(\mathbf{x}_1) \end{bmatrix}$, and

$$\mathbf{\Gamma}^*(\mathbf{u}_c, \mathbf{x}) = \begin{bmatrix} 0 \\ \mathbf{\Gamma}(\mathbf{u}_c, \mathbf{x}) \end{bmatrix}.$$

And then, define a state observer (using a hat to denote estimated or learned values):

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\hat{\mathbf{d}} + \mathbf{K}(\mathbf{x} - \hat{\mathbf{x}}) + \hat{\mathbf{\Gamma}}^*(\mathbf{u}_c, \hat{\mathbf{x}}), \tag{4.6}$$

which can be written as:

$$\begin{bmatrix} \dot{\hat{\mathbf{x}}}_1 \\ \dot{\hat{\mathbf{x}}}_2 \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{M}^{-1}(\mathbf{x}_1) \end{bmatrix} \hat{\mathbf{d}} + \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 - \hat{\mathbf{x}}_1 \\ \mathbf{x}_2 - \hat{\mathbf{x}}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \hat{\mathbf{\Gamma}}(\mathbf{u}_c, \mathbf{x}) \end{bmatrix},$$

Or, separating the two equations:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}_1 &= \hat{\mathbf{x}}_2 + K_{11}(\mathbf{x_1} - \hat{\mathbf{x_1}}) + K_{12}(\mathbf{x_2} - \hat{\mathbf{x_2}}) \\ \dot{\hat{\mathbf{x}}}_2 &= \mathbf{M}^{-1}(\mathbf{x_1})\hat{\mathbf{d}} + K_{21}(\mathbf{x_1} - \hat{\mathbf{x_1}}) + K_{22}(\mathbf{x_2} - \hat{\mathbf{x_2}}) + \hat{\mathbf{\Gamma}}(\mathbf{u}_c, \mathbf{x}), \end{aligned}$$

where $\hat{\mathbf{\Gamma}}(\mathbf{u_c}, \hat{\mathbf{x}})$ is the estimation of $\mathbf{\Gamma}(\mathbf{u_c}, \mathbf{x})$, computed as defined in Eq. (4.3), with the observed value of $\mathbf{x}$:

$$\hat{\mathbf{\Gamma}}(\mathbf{x}_1, \hat{\mathbf{x}}_2) = \mathbf{M}^{-1}(\mathbf{x}_1)[\mathbf{u}_c - \mathbf{n}(\mathbf{x_1}, \hat{\mathbf{x}}_\mathbf{2})]. \tag{4.7}$$

From now on, we will use $\hat{\mathbf{\Gamma}} = \hat{\mathbf{\Gamma}}(\mathbf{u_c}, \hat{\mathbf{x}})$ and $\mathbf{\Gamma} = \mathbf{\Gamma}(\mathbf{u}_c, \mathbf{x})$.

We must remark that in [111], this last term in (4.7) is assumed to be known. However, using its learned value will not affect the error dynamics. In fact, if the state estimation error is $\mathbf{e} = \hat{\mathbf{x}} - \mathbf{x}$ and the disturbance estimation error $\mathbf{e}_d = \hat{\mathbf{d}} - \mathbf{d}$, the error dynamics, subtracting (4.5) from (4.6), is:

$$\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{K})\mathbf{e} + \mathbf{B}\mathbf{e}_d + \hat{\mathbf{\Gamma}}^* - \mathbf{\Gamma}^*, \tag{4.8}$$

where, if we define (following the steps in [111]):

$$\hat{\mathbf{d}} = \mathbf{F}_1\mathbf{x} + \mathbf{F}_2\dot{\mathbf{x}} + \mathbf{G}_1\hat{\mathbf{x}} + \mathbf{G}_2\dot{\hat{\mathbf{x}}} + \mathbf{G}_3\mathbf{\Gamma}^* \tag{4.9}$$

then, as $\mathbf{F}_2\mathbf{B} - I \neq 0 \; \forall \mathbf{F}_2$, [111] proposes to take:

$$\begin{aligned} \mathbf{G}_1 &= -(\mathbf{F}_1 + \mathbf{B}^\dagger\mathbf{A}) \\ \mathbf{G}_2 &= -(\mathbf{F}_2 - \mathbf{B}^\dagger) \\ \mathbf{G}_3 &= -\mathbf{B}^\dagger, \end{aligned}$$

thus

$$\hat{\mathbf{d}} = \mathbf{F}_1\mathbf{x} + \mathbf{F}_2\dot{\mathbf{x}} - (\mathbf{F}_1 + \mathbf{B}^\dagger\mathbf{A})\hat{\mathbf{x}} - (\mathbf{F}_2 - \mathbf{B}^\dagger)\dot{\hat{\mathbf{x}}} - \mathbf{B}^\dagger\hat{\mathbf{\Gamma}}^*. \tag{4.10}$$

From (4.5), we can isolate $\mathbf{d}$ as $\mathbf{B}$ is full column rank, using its pseudoinverse $\mathbf{B}^\dagger$:

$$\mathbf{d} = \mathbf{B}^\dagger\dot{\mathbf{x}} - \mathbf{B}^\dagger\mathbf{A}\mathbf{x} - \mathbf{B}^\dagger\mathbf{\Gamma}^*, \tag{4.11}$$

and, with (4.10) and (4.11), knowing that, in the case of study, $\mathbf{B}^\dagger\mathbf{A} = \mathbf{0}$:

$$\mathbf{e}_d = \hat{\mathbf{d}} - \mathbf{d} = (\mathbf{B}^\dagger - \mathbf{F}_2)\dot{\mathbf{e}} - \mathbf{F}_1\mathbf{e} + \mathbf{B}^\dagger(\hat{\mathbf{\Gamma}}^* - \mathbf{\Gamma}^*), \tag{4.12}$$

where $\mathbf{B}^\dagger(\hat{\mathbf{\Gamma}}^* - \mathbf{\Gamma}^*) = \mathbf{n}(\mathbf{q}, \hat{\mathbf{x}}_\mathbf{2}) - \hat{\mathbf{n}}(\mathbf{q}, \hat{\mathbf{x}}_\mathbf{2})$ is the error with the learned model of the function $\mathbf{n}$ defined before. This means that, as one could expect, the force estimation error will depend on:

- The model estimation error.
- The estimated joint acceleration error.
- The estimated joint velocity error.

Thus, our objective will be to have a small-as-possible estimation error for the state space, to reduce the force estimation error $\mathbf{e}_d$.

Substituting (4.12) into (4.8), the $\mathbf{\Gamma}$'s cancel out and we obtain the position estimation error dynamics equation:

$$(I + \mathbf{B}\mathbf{F}_2 - \mathbf{B}\mathbf{B}^\dagger)\dot{\mathbf{e}} = (\mathbf{A} - \mathbf{K} - \mathbf{B}\mathbf{F}_1)\mathbf{e}.$$

Now, as we intend not to use acceleration measures in Eq. (4.11), we need $\mathbf{F}_2 = \begin{bmatrix} 0 & 0 \end{bmatrix}$, and with $\mathbf{F}_1 = \begin{bmatrix} 0 & \mathbf{M}(\mathbf{x}_1)\mathbf{\Sigma} \end{bmatrix}$, $\mathbf{\Sigma}$ being another gain, we obtain:

$$\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \dot{\mathbf{e}} = \begin{bmatrix} -\mathbf{K}_{11} & I - \mathbf{K}_{12} \\ -\mathbf{K}_{21} & -\mathbf{\Sigma} - \mathbf{K}_{22} \end{bmatrix} \mathbf{e}, \tag{4.13}$$

which can be operated to obtain the system:

$$\begin{aligned} \dot{\mathbf{e}}_1 &= -\mathbf{K}_{11}\mathbf{e}_1 + (I - \mathbf{K}_{12})\mathbf{e}_2 \\ \mathbf{e}_2 &= -(\mathbf{\Sigma} + \mathbf{K}_{22})^{-1}\mathbf{K}_{21}\mathbf{e}_1 \end{aligned},$$

thus $\mathbf{e}_2$ can be substituted in the first equation to obtain $\mathbf{e_1}$'s dynamics, the dynamics of the position estimation error:

$$\dot{\mathbf{e}}_1 = -\left[\mathbf{K}_{11} + (I - \mathbf{K}_{12})(\mathbf{\Sigma} + \mathbf{K}_{22})^{-1}\mathbf{K}_{21}\right]\mathbf{e}_1, \tag{4.14}$$

which converges for any values of K_{ij} and $\mathbf{\Sigma}$ for which (4.14)'s matrix in brackets is positive definite.

In addition, we have a dependency between $\mathbf{e}_2$'s dynamics and $\mathbf{e_1}$'s, meaning that if the position estimation converges, so does the velocity estimation. Also, if (4.13) has an asymptotically stable equilibrium point at $\mathbf{e} = 0$, from (4.8) we have (at steady state)

$$\mathbf{e}_d = \mathbf{B}^{\dagger}\left(\hat{\mathbf{\Gamma}}^* - \mathbf{\Gamma}^*\right) = \hat{\mathbf{n}} - \mathbf{n},$$

which is the error of modeling the dynamics.

Moreover, from (4.13) we have:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}_1 &= \hat{\mathbf{x}}_2 + \mathbf{K}_{11}(\mathbf{x_1} - \hat{\mathbf{x_1}}) + \mathbf{K}_{12}(\mathbf{x_2} - \hat{\mathbf{x}}_2) \\ 0 &= \mathbf{K}_{21}(\mathbf{x_1} - \hat{\mathbf{x_1}}) + (\mathbf{\Sigma} + \mathbf{K}_{22})(\mathbf{x_2} - \hat{\mathbf{x}}_2) \end{aligned},$$

which can be operated to get a linear dynamic equation for $\mathbf{x_1}$, $\mathbf{x_2}$:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}_1 &= \left[\mathbf{K}_{11} + (I - \mathbf{K}_{12})(\mathbf{\Sigma} + \mathbf{K}_{22})^{-1}\mathbf{K}_{21})\right](\mathbf{x_1} - \hat{\mathbf{x}}_1) + \mathbf{x_2} \\ \hat{\mathbf{x}}_2 &= (\mathbf{\Sigma} + \mathbf{K}_{22})^{-1}\mathbf{K}_{21}(\mathbf{x_1} - \hat{\mathbf{x}}_1) + \mathbf{x_2} \end{aligned},$$

seeing $\mathbf{x_1}$, $\mathbf{x_2}$ as inputs of the observer, this results in a dynamic system on $\hat{\mathbf{x}}_1$, being $\hat{\mathbf{x}}_2$ an output.

Finally, the external torque estimation (using equation (4.9) and our proposed values) is:

$$\hat{\mathbf{d}} = \mathbf{M}(\hat{\mathbf{x}}_\mathbf{1}) \left(\dot{\hat{\mathbf{x}}}_\mathbf{2} + \mathbf{\Sigma}(\mathbf{x_2} - \hat{\mathbf{x}}_\mathbf{2}) \right) + \hat{\mathbf{n}}(\hat{\mathbf{x}}_\mathbf{1}, \hat{\mathbf{x}}_\mathbf{2}) - \mathbf{u_c}. \tag{4.15}$$

This latter method is the first approach at applying [111] to a robotic manipulator, but it should be noted that:

- The approximate value of $\mathbf{\Gamma}$ in Eq. (4.10) would turn into noise on the observer, but as it is canceled out in (4.15), it is not supposed to affect the convergence of the state observer, although it indeed includes error in the contact force estimation.
- Criteria on the tuning of $\mathbf{K}$ are given.
- We require the use of the true joint velocities, $\mathbf{x}_2$. This is not a problem, as these can be measured by differentiating joint positions.
- No assumptions on the disturbance behavior or model are taken, except that $\mathbf{n}$ depends only on position and acceleration variables. Here it must be pointed that most disturbance observers in literature assume the disturbances have a *Lipschitz* behavior [112], or a known model [113]. Also, no steady-state requirements are needed, although the model may have more error.
- This estimation is independent of the control scheme used. It can be run online and parallel to any controller, even at a different frequency. However, as we will see later, it does become control dependent in certain situations due to unmodelled static friction and other unlearnable effects.

As a conclusion, the estimation of the disturbance, which is the external force, can be done with guarantees of convergence.

4.1.2 Experimentation

To test the observer proposed in Section 4.1.1, we implemented the previous equations on a 7-DoF WAM robot. As a control law, we used a computed torque control scheme [114] [77], with $\mathbf{u_c} = \mathbf{M}(\mathbf{x_1})\dot{\mathbf{x}}_\mathbf{2}^\mathbf{d} + \hat{\mathbf{n}}(\mathbf{x}_\mathbf{1}^\mathbf{d}, \mathbf{x}_\mathbf{2}^\mathbf{d}) + \mathbf{u_{PD}}$, $\hat{\mathbf{n}}(\mathbf{x}_\mathbf{1}^\mathbf{d}, \mathbf{x}_\mathbf{2}^\mathbf{d})$ being the learned model using the desired trajectory, instead of real measurement, and $\mathbf{u_{PD}}$ a PD control action on the joint state that compensates modeling error and any external force. Using it in the robot's dynamic equation, Eq. (2.10), we obtain:

$$\mathbf{M}(\mathbf{x_1})\dot{\mathbf{x}}_\mathbf{2} + \mathbf{n}(\mathbf{x_1}, \mathbf{x_2}) = \mathbf{M}(\mathbf{x_1})\dot{\mathbf{x}}_\mathbf{2}^\mathbf{d} + \hat{\mathbf{n}}(\mathbf{x}_\mathbf{1}^\mathbf{d}, \mathbf{x}_\mathbf{2}^\mathbf{d}) + \mathbf{u_{PD}} - \mathbf{u_e},$$

and, isolating $\mathbf{u_e}$, the real disturbance value is:

$$\mathbf{u_e} = \mathbf{u_{PD}} + \mathbf{M}(\mathbf{x_1}) \left(\dot{\mathbf{x}}_\mathbf{2}^\mathbf{d} - \dot{\mathbf{x}}_\mathbf{2} \right) + \hat{\mathbf{n}}(\mathbf{x}_\mathbf{1}^\mathbf{d}, \mathbf{x}_\mathbf{2}^\mathbf{d}) - \mathbf{n}(\mathbf{x_1}, \mathbf{x_2}).$$

However, with Eq. (4.15), we can substitute the control action $\mathbf{u_c}$ to obtain the estimated external perturbation:

$$\begin{aligned} \hat{\mathbf{u}}_\mathbf{e} = -\hat{\mathbf{d}} = \quad & \mathbf{u_{PD}} + \mathbf{M}(\mathbf{x_1}) \left(\mathbf{\Sigma}(\hat{\mathbf{x}}_\mathbf{2} - \mathbf{x_2}) + \dot{\mathbf{x}}_\mathbf{2}^\mathbf{d} - \dot{\hat{\mathbf{x}}}_\mathbf{2} \right) + \\ & + \hat{\mathbf{n}}(\mathbf{x}_\mathbf{1}^\mathbf{d}, \mathbf{x}_\mathbf{2}^\mathbf{d}) - \hat{\mathbf{n}}(\mathbf{x_1}, \hat{\mathbf{x}}_\mathbf{2}), \end{aligned}$$

where the real acceleration $\dot{\mathbf{x}}_\mathbf{2}$ is not used, but the estimated $\dot{\hat{\mathbf{x}}}_\mathbf{2}$ and desired $\dot{\mathbf{x}}_\mathbf{2}^\mathbf{d}$ ones are instead.

This system was discretized, and then the state observer system was run at 500Hz, while an inverse dynamic model learned with LWPR introduced in Section 2.3.1, was used and run, at 50Hz with a zero-order hold to attach it to the other 500Hz system. However, although simulation showed excellent results, after implementing the algorithms in a real robot, we found 3 factors that had to be mitigated in order to have better results. They are described in the following three subsections.

Noise

The joints position signal presented little noise, but differentiating in order to get the velocity increased noise. Moreover, when derivating $\hat{\mathbf{x}}_\mathbf{2}$ in order to get the estimated acceleration, a very noisy signal was obtained. In order to reduce this noise, which would directly affect the external torque result, we added a Parks-McClellan filter [82] at the readings of the joint state and velocity. With this filter, the estimated acceleration was less noisy than the obtained by directly differentiating the position readings twice.

Friction

The WAM robot is driven by cables in an architecture designed to reduce friction. However, when working with small controller gains and small velocities, the motor cogging and static friction become very evident. Static friction causes unpredicted stationary errors when the robot stops, and motor cogging adds a variable hysteresis on friction that makes model estimation difficult and causes a discontinuous tracking with the lowest inertia joints. As this friction cannot be learned, our work has focused on compensating the error they cause.

Error

The residual error, higher than expected, caused by unlearned static friction, results in a large PD action, where it should be small. The PD action, multiplied by the error, may give fake external torques in Eq. (4.15) if its constants are not small, i.e., large controller gains give larger force estimation errors for the same position error. This increases the importance of developing a friction model, as we will see in Section 4.2.

Experimental setup and results

To evaluate the behavior of the estimator, we trained a 10 seconds trajectory to a WAM robot, hanging different loads at its end-effector. The loads were of 0, 0.5, 1, 1.5 and 2 kg. To analyze the results, we compared the outputs of the vertical force estimated (F_z) at each trajectory, and the results are shown in Fig. 4.2, where we see that the estimation has low error, but accumulates slight error for large weights. This is because, as commented, a heavier load results in more error at stationary state, which implies a larger gain of the controller, thus more uncertainty in the estimation.

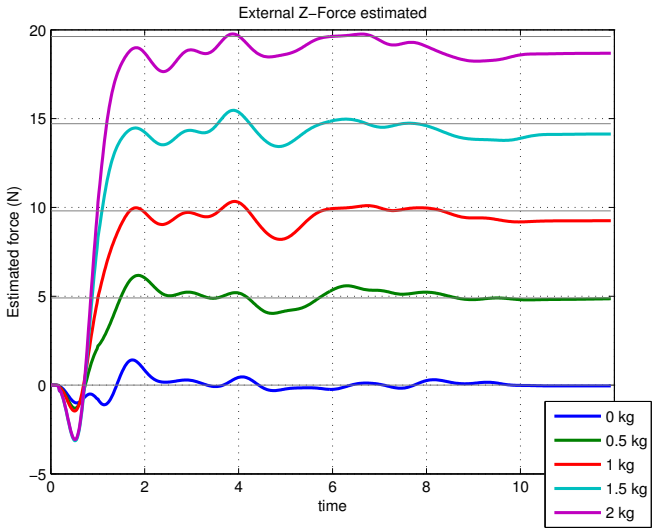


Figure 4.2: Experimental results when hanging weights from the robot's end-effector. Horizontal lines represent the real weight. At time$=10s$, the robot ended its trajectory. At time$=5s$, joint 1 changes its direction with a step on the desired acceleration, thus creating a transient in force estimation.

In Fig. 4.3 we can see the estimated torques along the trajectory, while in Fig. 4.4 we plot the resulting wrench estimations. There we can observe unexpected peaks due to joint 1 (with the most inertia) changing its direction. Static friction appears in that moment, causing the observed

behavior. This results in an unexpected transient estimated force, that rapidly decreases to zero afterwards. In addition, unmodelled frictions compensate part of the weight, thus the force estimation is slightly lower than the true weights on the end-effector.

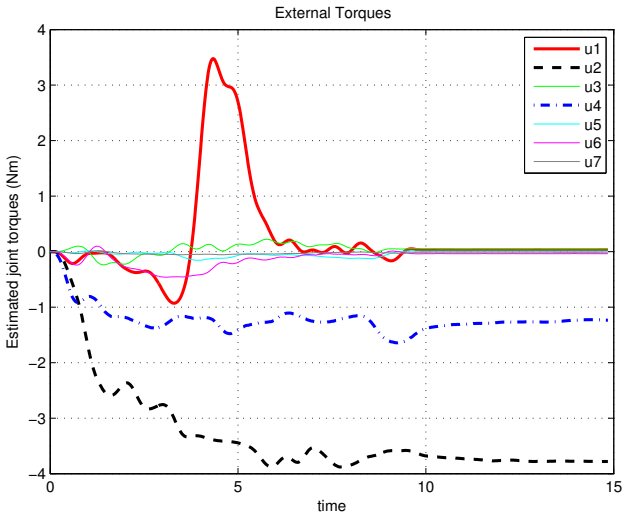


Figure 4.3: Estimated external torques for a load of $1kg$. Along the executed trajectory, joints 2 and 4 hold the vertical force. The transient peak on joint at time=5s is due to a step on the desired acceleration along trajectory.

The results show that unmodelled friction reduces the precision of our external force estimator. However, despite the uncertainties around unmodelled forces, the results in Fig. 4.2 and Fig. 4.4 are accurate, showing the potential use for any robot.

4.1.3 Discussion

Estimating the external force applied to a robot without having an expensive force sensor at its end-effector is a step forward for control and manipulation purposes. Some works have good results in simulation, but rely on the availability of analytical models of the robot, the possibility of having the true values of friction or Coriolis forces or they assume almost-stationary situations, meaning the estimations are not available while the robot moves. In this section, we have proposed an algorithm that, despite its estimation may have a small delay caused by the filters applied and can carry errors due to unknown friction, outperforms the previous works based on state observers, with a rigorous deduction of equations and proof of its convergence, making no assumptions on the external torque applied, nor requiring stationary situations.

The results, in Section 4.1.2, with a 7-DoF robot capable of performing various manipulation tasks, show that good force estimates can be obtained while the robot is still in motion. These

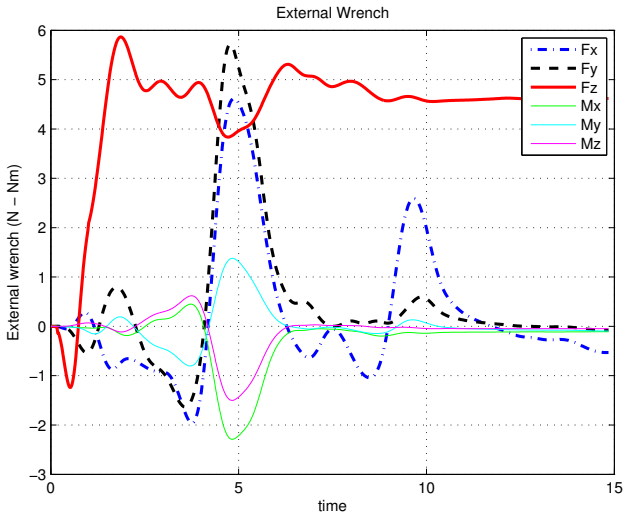


Figure 4.4: Estimated external wrenches for a load of $0.5kg$. Acceleration discontinuities when changing direction or stopping cause transient behavior at time=$5s$ and $10s$.

vertical force results are being used to know how many cloth garments have been picked by a robot. Another advantage of this proposal is that it can be implemented on any control scheme (see Fig. 4.1).

As future work, some research to reduce the effects of uncertainties on the dynamic behavior of the robot, which generate error, needs to be carried out. Other possible improvements are to globalize the inverse model to the whole workspace, instead of tailoring it to trajectories, and to optimize the robot controller in order to reduce the residual torques observed, or model friction apart so as to gain precision.

4.2 Building a friction model

Most approaches to build an Inverse Dynamics Model (IDM) do not consider the possibility of having hysteresis on the friction, which is the case for robots such as the Barrett WAM. For this reason, in order to improve the available IDM, we derived an analytical model of friction in the seven robot joints, whose parameters can be automatically tuned for each particular robot. This permits compliantly tracking diverse trajectories in the whole workspace.

By using such friction-aware controller, Dynamic Movement Primitives (DMP, see Chapter 5) as motion characterization and visual/force feedback within the RL algorithm, experimental results to be presented in Section 4.3 demonstrate that the robot is consistently capable of learning such safety-critical tasks.

4.2.1 Introduction

Not properly representing this hysteresis reduces the accuracy of the IDM, which implies a control error that requires increasing the error-compensating gain to ensure a good position tracking, thus making the robot less compliant. For this reason, in Section 2.3.2 we proposed an analytical formulation of the friction, which can be easily tuned with real data and then computed online, providing good results in the whole joint space of the robot.

Once having this IDM, we built a learning framework (see Fig. 4.5), using a proper motion characterization and a model-based External Force Estimation (EFE) as defined in the previous section to obtain the interaction torques with the environment. Within this framework, starting from an initial demonstration, exploration on the DMP parameters progressively improves the robot's performance and, using a Policy Search algorithm, better robot behaviors are learned relying on a cost function based on vision, kinematics and dynamics.

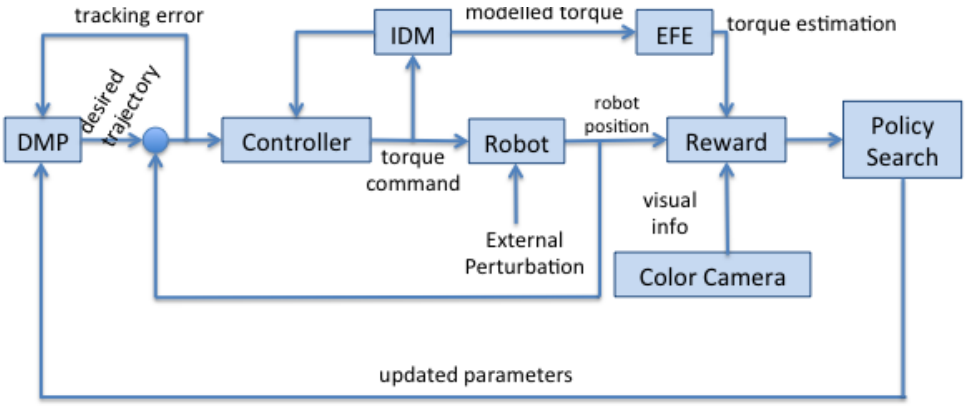


Figure 4.5: Global scheme of the proposed framework. The DMP sends the desired trajectory to the feed-forward controller, which uses the inverse dynamic model to track the trajectory compliantly. The robot performs the task, which may include an interaction with the environment, estimated by the external force estimator. The latter is used, together with the camera feedback and the acceleration measurements, to obtain a reward/cost function, which is used by a policy search algorithm to, after a certain number of rollouts, update the DMP.

4.2.2 Advanced model for the WAM robot

To improve the basic model some changes were made and new variables and terms were added:

- A linear term in position, which was observed through data analysis.
- A basis of Fourier functions on the joints position, divided in two layers, depending on the sign of the velocity of each joint.

These Fourier basis functions were added to model an oscillating curve wrt. the position of the friction. However, the authors observed different oscillations for positive and negative velocities, thus two sets of Fourier basis were fitted for them, and activated with the help of a sign function, as we can see in Eq. (4.16). These Fourier terms approximate the oscillations seen wrt. position, ignoring the noise data due to the PID controller that was used to obtain the data.

This brought us to a friction model for each of the joints of the robot with the following expression:

$$\begin{aligned} &F_f^i = b_1 q_i + b_2 \dot{q}_i + b_3 atan(s\dot{q}_i + z sign(\ddot{q}_i)) \\ &+0.5(1 + sign(\dot{q}_i))(b_4 f_1 + b_5 f_2 + b_6 f_3 + b_7 f_4 + b_8 f_5) \\ &+0.5(1 - sign(\dot{q}_i))(b_9 f_1 + b_{10} f_2 + b_{11} f_3 + b_{12} f_4 + b_{13} f_5), \\ &\text{for i} = 1..7, \end{aligned} \tag{4.16}$$

where $f_j = \frac{4}{\pi} \frac{sin((2j-1)hq_i)}{2j-1}$, $j = 1..5$ are the sine Fourier basis functions. We neglected the cosine functions assuming an antisymmetric behavior of the friction. q_i, $\dot{q}_i$ i $\ddot{q}_i$ are the position, the velocity and the acceleration of the ith joint. All the constraints b_k, $k = 1..13$, s, z i h were obtained with least-squares techniques and vary according to each joint.

This friction model has a high degree of accuracy only when joints were moving separately and did not take into account that some of the joints shared part of their friction due to the coupling between their engines.

Joints 1, 4 and 7 do no present this phenomenon, but joints 2-3 and joints 5-6 do when they move together in pairs. In order to model the friction torque of a pair of joints that share the coupling effect when they move we had to find a new model so that the friction torque was fitted with enough accuracy when this happened.

In this case, after several observation experiments, it could be seen that the friction torque applied on a joint was reduced when the other joint of the pair moved at a higher velocity. This reduced friction torque showed the same non-linear hysteresis phenomenon but with a smaller range of values and was modeled with the same expression but with different least squares parameters.

After modeling this reduced friction, a transition between models had to be established so that the friction torque was fitted correctly at every timestep. Defining $modA_i$ as the friction model of the ith joint when it moves at a higher velocity than its pair j (non reduced friction torque), $modB_i$ as the friction model of the same ith joint when it moves at a lower velocity than its pair j (reduced friction torque) and defining $mod_i(t)$ the friction model of the ith joint at timestep t:

$$mod_i(t) = modA_i(t), \text{ if } |\dot{q}_i(t)| \geq |\dot{q}_j(t)| + \epsilon$$
$$mod_i(t) = modB_i(t), \text{ if } |\dot{q}_j(t)| \geq |\dot{q}_i(t)| + \epsilon$$
$$mod_i(t) = mod_i(t-1), \text{ otherwise,}$$

for the pairs $(i = 2, j = 3)$, $(i = 3, j = 2)$, $(i = 5, j = 6)$ and $(i = 6, j = 5)$, where ϵ is used to avoid chattering between the friction models of each one of the coupled joints.

After defining this transition when the coupling phenomenon appears, the dynamic friction torques of each of the joints of the robot are modeled with enough accuracy, even if all the joints move together at the same time.

Fitting performance

In order to compare the performance of the different models in equations (2.13), (2.14) and (4.16), we fitted the three models with a dataset of 80 oscillation movements at different speeds, moving joints individually at different positions and also with coupled movements.

We built another dataset to validate the models and our proposed friction model in Eq. (4.16) showed to perform the best in all cases. In Fig. 4.6, we can see an example validation trajectory with the data obtained from Eq. (2.12). Also, in Table 4.1, we show the error indicators used in order to validate the advanced model against the initial and basic model with the new validation dataset. The error indicators used were the Mean Absolute Error (MAE), the Mean Squared Error (MSE) and the maximum sample error in the dataset. It can be clearly seen that this approach outperforms previous ones in terms of numerical error.

Table 4.1: Friction Validation Results

	Initial Model	Basic model	Adv. model
MAE	0.5907	0.5115	0.4277
MSE	0.7613	0.5611	0.3511
Max error	4.7281	3.6624	2.5849

Performance indicators for a set of N validation trajectories for the Barrett WAM joint 1. We indicate the MAE, MSE and maximum error, all of them averaged over 8 validation trajectories.

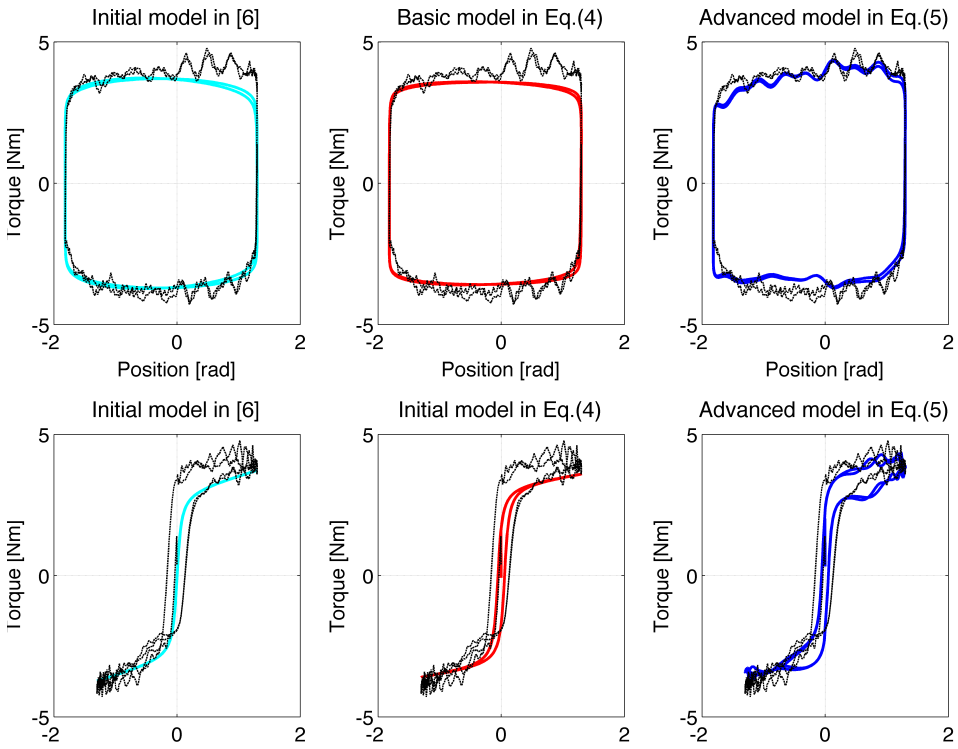


Figure 4.6: Example of the fitting of the three models for a trajectory of the first joint of the Barrett WAM. Our proposed function outperforms the previous approaches.

Feed-forward controller

Once given a desired trajectory, provided by the DMP or a similar framework, we need a proper controller to track it in a way that the robot is not dangerous. The default controller provided with the WAM robot is a PID controller with large gains: $\mathbf{u}_c = \mathbf{K}_p\mathbf{e}+\mathbf{K}_v\dot{\mathbf{e}}+\mathbf{K}_I\int_{\tau=0}^{\tau=t}\mathbf{e}_\tau$. However, as already mentioned before, a pure PID controller may not be suitable to interact with humans or deformable objects, thus we needed to implement another controller.

In order to properly track the reference trajectories generated by the DMPs, we include a Computed Torque Controller [77], in which we add an Inverse Dynamic Model (IDM), that allows us to use a low-gain error-compensating term, i.e.: we reduce the stiffness of the robot while performing the motion.

$\mathbf{u}_c = \mathbf{u}_{PD}+\mathbf{u}_{IDM}$ where $\mathbf{u}_{PD} = \mathbf{K}_p\mathbf{e}_p+\mathbf{K}_v\dot{\mathbf{e}}_p$ is a PD controller and the IDM is approximated by

$$\mathbf{u}_{IDM} \simeq \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\dot{\mathbf{q}},\mathbf{q}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}_f. \tag{4.17}$$

Using this controller, the robot will have a much more compliant, or *soft* behavior, as the IDM would just provide the necessary torque to follow the desired commands, while the PD part of the controller takes care of error compensation. In fact, the PD torque only acts to compensate model errors and external perturbations.

4.3 Applications

The compliant controller with external force estimation presented throughout this chapter has been implemented and extensively used along this thesis with the Barrett WAM robots available. Here, we show two example applications of its use.

4.3.1 Scarf-placing experiment

As an experimental setup for the scheme in Fig. 4.5, we decided to put a scarf to a boy-sized mannequin, placed at a fixed position wrt. the robot, and use a color camera to check if the scarf was properly placed after each rollout.

Initialization and cost function To that purpose, we initialized the DMP with 25 Gaussian kernels per degree-of-freedom, fitting a poor-performing motion which does not achieve such a task, and improve it over 20 epochs of 12 rollouts each. Using the CTC defined previously, which includes the friction model in Section 4.2, we reproduced the motions and obtained the costs by evaluating the cost function. The PI2 algorithm, with the CMA and a dual layer of Gaussians as proposed in [25], with 8 kernels per degree-of-freedom, was then used to update the policy.

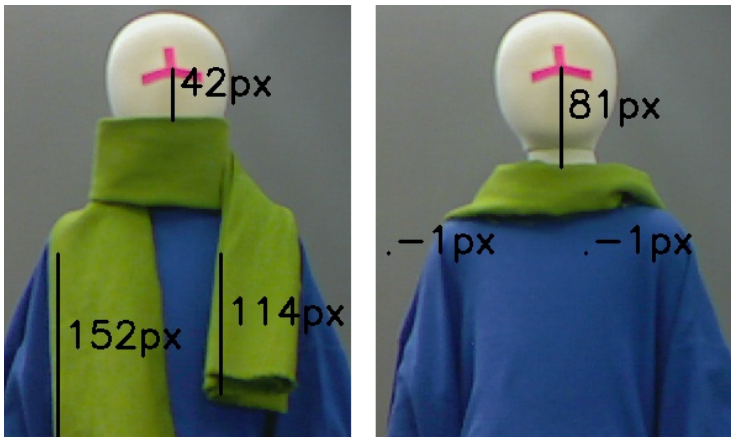


Figure 4.7: Two examples of the camera output, measuring the distance (in pixels) from the marker to the hanging scarf, and the length of the part hanging on the sides up to a certain level.

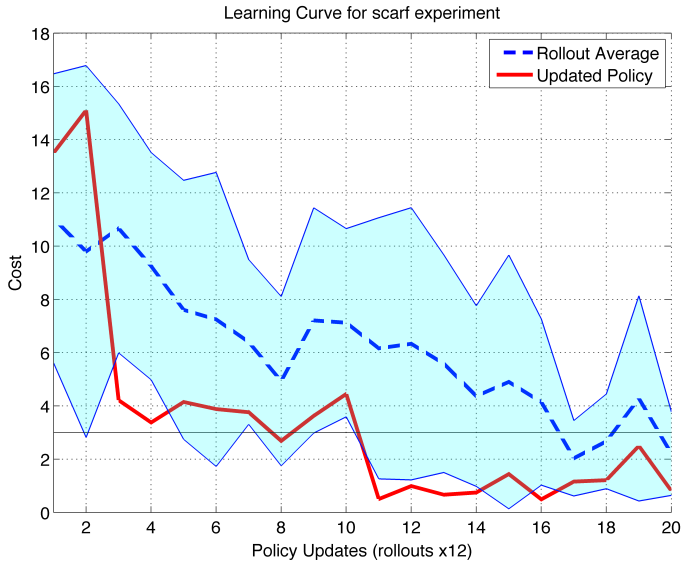


Figure 4.8: Learning curve for the scarf experiment. The red continuous line shows the exploration-free policy cost after each update, while the blue shaded area shows the mean and standard deviation for all the exploration rollouts at each epoch. The horizontal black line represents the cost value of 3, which is considered by the authors to be the threshold indicating whether the task was successfully completed or not.

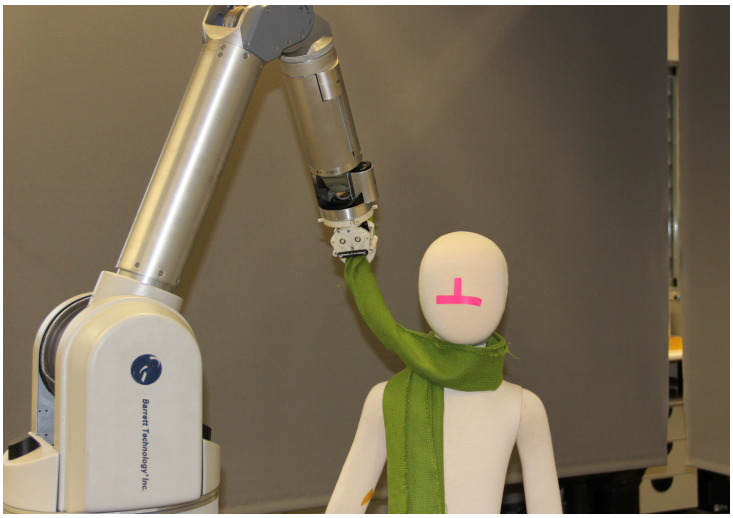


Figure 4.9: WAM after placing the scarf around the mannequin's neck.

The variance for exploration was initialized with a value $\Sigma_\Theta = 10I_{8 \cdot dof}$ and updated with a filtered version of the CMA algorithm in [115], keeping only the diagonal part of the matrix for simplicity.

As a reward function to optimize, we use a timestep cost c_t, $t = 1..T$ and a final cost c_T, which depend on the acceleration, interaction torques and visual feedback.

The total cost for a trajectory is then $C_T = c_T + \sum_{t=1}^{T} c_t$, where c_t is a timestep cost given by a quadratic form of the acceleration commands sent to the robot by the DMP, and the terminal cost $c_t = c_{cam} + c_{torque}$ is the sum of the costs given by the color camera and the EFE at the end of the rollout.

Desired acceleration: to penalize those task attempts where the policy (DMP parameters) tells the robot to move with high acceleration, we added a penalizing term to the cost function, consisting of a quadratic form on the acceleration at each timestep of the trajectory.

External force estimation: in order to punish those motions in which the robot tries to push or pull the scarf and/or mannequin too much, we estimate the external torques resulting from the interaction of the robot and the scarf, which will also include the transferred torque from the scarf-mannequin iteration to the robot at each timestep. For simplicity, we used the average of the absolute value of the estimated interaction torques all through the motion.

Visual feedback: at the end of each rollout, we use a color camera to check if the scarf was properly placed by the robot. Using HSV color segmentation to distinguish the colors with the code provided in [116], such as the mannequin color, scarf, and a mark placed on its nose. As a descriptive element, we used the distance d from the nose of the mannequin to the color-segmented scarf, in the vertical line from the nose. If the scarf is well-placed, this distance will be

close to a reference value d_{ref}. Otherwise, the scarf might be hanging too close to the nose or not covering the neck. We defined the scarf position cost as $c_{cam} = 10 \max\left((d - d_{ref})/d_{ref}, 1\right)^{\alpha} + 3I_{hanging}$, for a given exponent α (a value of 1.2 was used throughout this chapter), and a penalizing factor with the indicator function $I_{hanging}$, which is 1 if the scarf is not hanging on the right side of the mannequin or it is hanging in its left side, assuming we want the scarf only to be hanging on the mannequin's right side (see Fig. 4.7). In this case, the reference distance d_{ref} was 62 pixels for the color camera placed at a $1m$ distance from the mannequin.

This visual feedback cost is very rudimentary and was implemented only to demonstrate the performance of the entire system. Of course it can be replaced by most elaborate task-dependent cost measures.

Results In Fig. 4.8, we show the learning curve of the scarf-placing task, in which we can consider that a cost below 3 represents a motion that effectively placed the scarf around the mannequin's neck, hanging on the right side but not on the left side (see Fig. 4.9). We observe that, after 10 policy updates (i.e., 120 rollouts), the task has already been learned and, after that, the policy gradually reduces its variance, despite not being refined further, mainly due to all the uncertainties in the process of manipulating clothes. Other similar experiments have been done and can be seen in Appendix B.2.

4.3.2 Compliant object tracking

In [20], an automated system able to track a moving object with range images obtained with a Microsoft Kinect sensor is presented. Then, a robotic manipulator tracks such objects in real time, using the compliant controller defined in this chapter, together with the IK algorithm defined in Section 3.1.5 implemented in a real robot, following the scheme in Fig. 4.10. After tracking an object for a certain period of time, the robot is allowed to grasp it, as seen in Fig. 4.11. More information on this experiment, as well as videos of the robot tracking objects can be found on Appendix B.3.

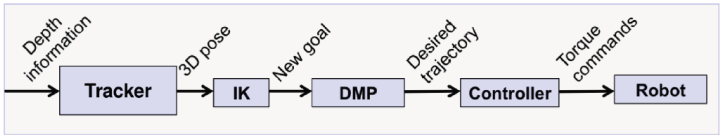


Figure 4.10: Tracking scheme for following a moving object with a WAM robot.

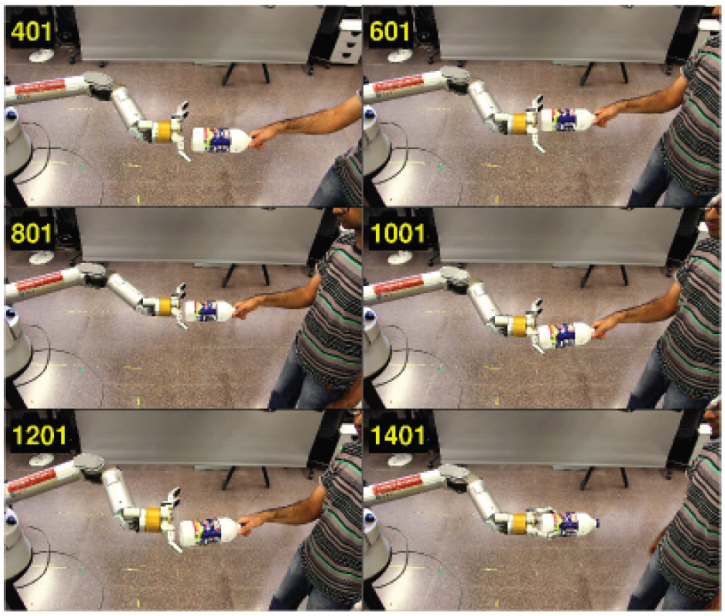


Figure 4.11: WAM robot tracking and grasping an object.

4.4 Summary

In Section 4.1, an External Force Estimation based on a disturbance observer has been proposed which, by using a previously learned dynamic model, estimates the external forces on the robot with the kinematics data and the control commands sent to the robot. This can be useful not only to predict interaction with the environment or detecting whether the robot is holding something, but also to try to minimize the interaction forces between the robot and its environment so as to reduce the stress on the manipulated objects.

In this chapter, we aimed at developing a compliant controller that ensures human safety in physical interaction tasks involving deformable objects. To that purpose, we developed a new friction model that is well suited for the case of the Barrett WAM robot and does not require as many samples as other IDM fitting approaches [78]. The IDM resulting from that friction model proved to be precise enough to compliantly but precisely track reference commands with a Computed Torque Controller (CTC), while also detecting contact forces.

Finally, we used a combination of kinematic, dynamic and visual feedback within a cost function in order to take all possible factors into account when learning a robotic task: Accelerations telling how smooth the desired trajectory was, estimated contact torques evaluating if there was a too hard interaction with the mannequin, probably due to the robot hitting or wrongly pushing it, and also a simple color-based segmentation that efficiently measured whether the robot was successful at the performed task. The difficulties of simulating deformable objects and human

environment force RL algorithms to require all these elements to be successful but safe while learning, as concluded by J. Kober in [117]. In this work, we provided an initial framework to safely learn tasks involving deformable objects in close proximity to humans.

As already introduced, this framework permits learning in compliant environments, as we will develop further in the second part of this thesis.

Part II

Reinforcement Learning with Movement Primitives

5
Preliminaries

In this part of the thesis, we used the kinematic and control architectures built in Part I in order to learn cooperative manipulation. To this endeavor, we will firstly introduce Policy Search (PS), a subtype of Reinforcement Learning in Section 5.1. For further details on PS, [12] presents a more exhaustive review, with a detailed description of many of the state-of-the-art PS algorithms and the reasoning behind them. We will also introduce the concept of Movement Primitives in Section 5.2, a motion characterization very suitable to PS.

5.1 Policy Search (PS)

In a wide variety of robotics applications, there exists a tradeoff between implementing tasks by careful engineering, or by letting the robot learn them, either from scratch or from a demonstrated initial approach. In engineering a task, hard-coding a robot's behavior can yield satisfactory results for repetitive applications such as some production chain industrial processes. Those solutions are usually difficult to generalize and, thus, more engineering is required every time an element of the task changes. For this reason, it is sometimes more desirable to implement a self-learning algorithm so the robot is capable of learning by itself and improving over new experiences. Consequently, robot control can be represented as a Reinforcement Learning (RL) problem [12], where the robot is supposed to take actions in an environment so as to maximize the expected value of a certain cumulative reward [3].

The standard RL can be defined with a Markov Decision Process (MDP), where the robot selects an action $\mathbf{a}$ in a given state $\mathbf{y}$, according to a policy π, which maximizes a reward function $\mathbf{R}$, provided that there is a transition probability of being in state $\mathbf{y}_{t+1}$ after applying action $\mathbf{a}_t$ when in state $\mathbf{y}_t$, $p(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{a}_t)$. MDPs are built with the hypothesis that the actions and states are finite sets. In the field of action planning, such structure can be easily defined as a sequence of selected behaviors given descriptive state variables. However, the application of such MDP

structures into robot control for motion learning is not as easy, due to the fact that the states $\mathbf{y}$, usually representing positions, velocities, etc. of robot's joints or end-effector's pose are high-dimensional and continuous spaces. This results in an infinite set of possible states, as well as actions. A different representation of states and actions is therefore needed.

5.1.1 Robot control as a reinforcement learning problem

For robot motion control, actions, formerly denoted $\mathbf{a}$, will encode the motor commands $\mathbf{u}$, and the state variables will be denoted $\mathbf{y}$, representing the robot's joint positions, end-effector pose, contextual variables, etc. Then, the control policy π will determine which motor command to apply in every state $\mathbf{y}$. Such policy π can be either stochastic $\pi(\mathbf{u}|\mathbf{y})$ or deterministic $\mathbf{u} = \pi(\mathbf{y})$. The motor commands $\mathbf{u}$ modify the robot's state with a transition probability $p(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{u}_t)$, while the concatenation of states and actions can be called a trajectory or rollout, and is represented as $\boldsymbol{\tau} = \{\mathbf{y}_1, \mathbf{u}_1, ..., \mathbf{y}_{N_t}, \mathbf{u}_{N_t}\}$. Additionally, we can assume the existence of a reward function, that will be an indicator of a trajectory quality:

$$\mathbf{R}(\boldsymbol{\tau}) = r_T(\mathbf{y}_t) + \sum_{t=0}^{T-1} r_t(\mathbf{y}_t, \mathbf{u}_t), \tag{5.1}$$

where r_t is the instantaneous reward at timestep t and r_T (T being the trajectory horizon) is a final reward. Then, RL will try to find the policy π that maximizes the expected reward:

$$J_\pi = \mathbb{E}\left[\mathbf{R}(\boldsymbol{\tau})|\,\pi\right] = \int \mathbf{R}(\boldsymbol{\tau}) p_\pi(\boldsymbol{\tau}) d\boldsymbol{\tau}, \tag{5.2}$$

where $p_\pi(\boldsymbol{\tau})$ is the probability of obtaining the trajectory $\boldsymbol{\tau}$ under the policy π.

This framework permits formulating robot control as a RL problem. However, there are additional difficulties in applying such representation for robot control. Formally, the biggest one is the fact that trajectories' states $\mathbf{y}$ and control actions $\mathbf{u}$ are usually continuous variables in an already high-dimensional space. This results in a high difficulty for using some of the most used RL approaches, like value function - based approaches. For these reasons, Policy Search (PS) methods use parametrized policies $\pi_{\boldsymbol{\theta}}$ which operate in a parameter space $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, thus the expected return to optimize in Eq. (5.2) becomes:

$$\mathbf{J}_{\pi_{\boldsymbol{\theta}}} = \mathbf{J}_{\boldsymbol{\theta}} = \mathbb{E}\left[\mathbf{R}(\boldsymbol{\tau})|\,\boldsymbol{\theta}\right] = \int \mathbf{R}(\boldsymbol{\tau}) p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \tag{5.3}$$

Several PS algorithms have been used in Robotics to solve Eq. (5.3) over the last decade, and these are mainly separated as model-based or model-free PS algorithms [12]. Model-free algorithms learn policies directly from trajectory samples, while model-based algorithms firstly

learn a model of the task, to later improve the policy. In the scope of this thesis, only model-free approaches are used, as deformable objects such as cloth garments are difficult to model and, therefore, model-based approaches would need too many samples for learning a model.

In general, when using PS learning algorithms, some robot trajectories $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, ...$ are sampled from the current stochastic policy π, and the rewards $R_1, R_2, ...$ obtained with them are evaluated. After a number of samples, the trajectories and rewards are used by the PS to directly obtain a new policy π^{new}, parametrized by $\boldsymbol{\theta}$. This new policy is the one maximizing certain criterion related with the average return $J_{\boldsymbol{\theta}} = \mathbb{E}\left[R(\boldsymbol{\tau}|\boldsymbol{\theta})\right]$ in Eq. (5.3). In this thesis, we represent the stochastic policies by probability distributions over a set of parameters. A policy $\pi(\boldsymbol{\theta})$ is then represented by a normal distribution with mean $\boldsymbol{\mu}_\omega$ and covariance $\boldsymbol{\Sigma}_\omega$, i.e., $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega\}$, generating samples $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$. Model-free PS use such stochastic policies to generate new trajectories to be evaluated, and can be divided into three main approaches:

- Policy Gradients methods use gradient ascent to obtain a better average return $J_{\boldsymbol{\theta}}$ [118]. Therefore, the policy update rule becomes:

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}, \tag{5.4}$$

  where α is a learning rate. The main contribution of such methods is then how to properly estimate the policy gradient $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}} = \int_{\boldsymbol{\tau}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}} R(\boldsymbol{\tau}) d\tau$ from samples, so that the new policy is as good as possible. In [119], some of these methods are defined, many based on the *Likelihood-ratio* trick given by the property that $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}} = p_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}$, as well as the policy gradient theorem, which imposes that under a non-changing policy, future actions do not depend on past rewards. Other policy gradient approaches are based on optimal control theory, as in the case of Policy Improvement with Path Integrals (PI2).

- Expectation Maximization methods do not need a learning rate, which can sometimes be problematic. In contrast, EM-based PS considers the learning problem as an inference problem [12].

- Information Theoretic approaches try to stay close to the given data when computing the new policy. Robotic applications usually need to limit the policy variation from one iteration to another, in order to keep a safe robot behavior. Additionally, EM approaches usually present a too greedy behavior. Therefore, algorithms based on limiting the change in the policy by setting a bound on the Kullback-Leibler divergence [120] between the old and new policies have been appearing in the last decade, such as Relative Entropy Policy Search [121].

Now we briefly present two of the most popular PS algorithms found in literature: REPS and PI2, which have been used in the experiments throughout this thesis.

Relative Entropy Policy Search (REPS)

Formally, REPS [121, 122] finds the policy π^* that maximizes the expected reward for a given task. The REPS algorithm uses Kullback-Liebler (KL) divergence [120], which is a non-symmetric indicator of the difference between two probability distributions p, q over a random variable x:

$$\text{KL}(p\|q) = \int p(x)\log\frac{p(x)}{q(x)}dx \tag{5.5}$$

Given the previous policy $q(\boldsymbol{\theta})$, the new policy $\pi(\boldsymbol{\theta})$ is obtained by adding a KL-Divergence bound ϵ between the newly obtained policy and the previous one to the optimization of the expected reward. The bound on the KL-Divergence limits the variation on the new policy and prevents the PS algorithm from being too greedy. Too greedy algorithms can be a wrong approach in some robotics applications, where a drastic change in the policy may result in an unpredictable, dangerous behavior of the robot. Such new policy π^* is then computed as the solution of:

$$\begin{aligned} \pi^* &= \text{argmax}_\pi \int \pi(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta})d\boldsymbol{\theta} \\ \text{s.t. } \epsilon &\geq \text{KL}(\pi(\boldsymbol{\theta})\|q(\boldsymbol{\theta})) \\ 1 &= \int \pi(\boldsymbol{\theta})d\boldsymbol{\theta} \end{aligned} \tag{5.6}$$

where $\boldsymbol{\theta}$ are the parameters, $\mathbf{R}(\boldsymbol{\theta})$ is their associated reward, and $\pi(\boldsymbol{\theta})$ is a probability distribution over the parameter space.

Solving the constrained optimization problem in (5.6) provides a solution of the form [121]

$$\pi^* \propto q(\boldsymbol{\theta})\text{exp}(-\mathbf{R}(\boldsymbol{\theta})/\eta), \tag{5.7}$$

where η is the Lagrange multiplier of the KL bound and can be obtained by through numerical optimization.

Given the value of η and the rewards, the exponential part in (5.7) acts as a weight $d_k = \text{exp}(-R_k/\eta)$, with $R_k = \mathbf{R}(\boldsymbol{\theta}_k)$[1] to be used with the samples $\boldsymbol{\theta}_k$ in order to obtain the new policy, usually with a Gaussian Weighted Maximum Likelihood Estimation (WMLE). However, it has been shown [123] that the order of KL factors used in REPS - $\text{KL}(\pi(\boldsymbol{\theta})\|q(\boldsymbol{\theta}))$ instead of $\text{KL}(q(\boldsymbol{\theta})\|\pi(\boldsymbol{\theta}))$ - has an averaging-between-solutions behavior. In fact, using the reverse order would help to find a single solution faster, but there is no closed REPS solution using $KL(q\|\pi)$ instead of $KL(\pi\|q)$, as is more detailed in Chapter 6.

[1] For numerical reasons, $\frac{R_k - \min(\mathbf{R})}{\max(\mathbf{R}) - \min(\mathbf{R})}$ is often used instead.

Policy Improvement with Path Integrals (PI2)

PI2 [124] takes inspiration on the optimal control theory, defining the timestep reward term as $r_t = r_t(\mathbf{y_t}) + \mathbf{u_t}^T \mathbf{\Lambda} \mathbf{u_t}$, $\mathbf{u}_t$ being the control action, which includes a Gaussian noise with variance $\mathbf{\Sigma}_\epsilon$. Under the assumption that $\lambda \mathbf{\Lambda}^{-1} = \mathbf{\Sigma}_\epsilon$ for some real value λ, the Hamilton-Jacobi-Bellman equation of the optimal control problem can be solved [124]. Additionally, if the control actions are linearly parametrized with $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega)$, then multiplying a certain basis functions vector $\boldsymbol{\phi}_t$ as is the case of movement primitives, then, given a set of N_k sample trajectories $\boldsymbol{\tau}_1 ... \boldsymbol{\tau}_{N_k}$ with parameters $\boldsymbol{\omega}_1 ... \boldsymbol{\omega}_{N_k}$, we can rewrite the timestep reward as $r_t^k = r_t(\mathbf{y_t}) + \boldsymbol{\omega}_k^T \mathbf{\Lambda} \boldsymbol{\omega}_k$ and the new policy can be obtained with the following steps [124]:

- **step 1.** Generate N_k rollouts with parameters sampled from the policy π: $\boldsymbol{\omega}_k \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega)$, $k = 1..N_k$

- **step 2.** For all N_k rollouts, compute:

$$M_{t_j,k} = \frac{\mathbf{R}^{-1} \boldsymbol{\phi}_t \boldsymbol{\phi}_t^T}{\boldsymbol{\phi}_t^T \mathbf{R}^{-1} \boldsymbol{\phi}_t} \tag{5.8}$$

  And the cost for each sampled trajectory:

$$S(\boldsymbol{\tau}_{i,k}) = r_{N_t} + \sum_{j=i}^{N_t - 1} r_{t_j,k} + \frac{1}{2} \sum_{j=i+1}^{N_t - 1} (\boldsymbol{\mu}_\omega + M_{t_j,k} \boldsymbol{\omega})^T \mathbf{R}^{-1} (\boldsymbol{\mu}_\omega + M_{t_j,k} \boldsymbol{\omega}) \tag{5.9}$$

  As well as the associated timestep probabilities for each rollout and each timestep:

$$P(\boldsymbol{\tau}_{i,k}) = \frac{e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,k})}}{\int e^{-\frac{1}{\lambda} S(\boldsymbol{\tau}_{i,k})}} \tag{5.10}$$

- **step 3.** Compute timestep gradients

$$\delta \boldsymbol{\omega}_{t_i} = \sum_{k=1}^{N_k} P(\boldsymbol{\tau}_{i,k}) M_{t_j,k} (\boldsymbol{\omega} - \boldsymbol{\mu}_\omega) \tag{5.11}$$

- **step 4.** The policy gradient is finally

$$\delta \boldsymbol{\omega} = \frac{\sum_{i=0}^{N_k - 1} (N_k - i) \delta \boldsymbol{\omega}_{t_i}}{\sum_{i=0}^{N_k - 1} (N_k - i)} \tag{5.12}$$

- **step 5.** Update the policy

$$\boldsymbol{\mu}_\omega = \boldsymbol{\mu}_\omega + \delta\boldsymbol{\omega} \tag{5.13}$$

Such policy update does not take into account the exploration parameters, the covariance matrix $\boldsymbol{\Sigma}_\omega$. Therefore, other variants of PI2 like PI2 with Covariance Matrix Adaptation (CMA) [115] also use the weightings in Eq. (5.10) to update the covariance matrix, as well as evolutionary strategies.

These PS algorithms, when applied to robotic trajectory learning problems, require a proper motion representation. In the following section, we will define two of the most used Movement Primitives (MP) in literature.

5.2 Movement Primitives (MP)

In order to represent parametrized policies for applying PS algorithms in robotics, it is desirable to have characterizations with minimal expressions and number of parameters. To that purpose, if we have a robot trajectory $\boldsymbol{\tau} = \{\mathbf{y}_0, ..., \mathbf{y}_{N_t}\}$, the easiest way of storing such trajectory would be to store all its points and then reproduce them on a robot. However, this would result in too many parameters in order to efficiently use PS. Spline fitting might also be considered, but spline parameters are obtained by solving a set of equations, meaning that parameters are related to each other, and those are not usually very intuitive. Therefore, most used approaches to encode robot motion policies are Movement Primitives. In this chapter, we will present two of the most popular ones: Dynamic Movement Primitives (DMPs), based on second-order dynamic systems, and Probabilistic Movement Primitives (ProMPs), linear policies representing stochastic trajectories. While DMPs are a motion representation as a dynamical system linear with the parameters at an acceleration level, ProMPs are a probabilistic representation with its position linear wrt. policy parameters. Within these models, in order to generate samples for learning, trajectory policies will be stored with a parameter mean $\boldsymbol{\mu}_\omega$ and a covariance $\boldsymbol{\Sigma}_\omega$. Such $\boldsymbol{\Sigma}_\omega$ will be used to generate new samples $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$ that will be executed and evaluated.

5.2.1 Dynamic movement primitives

Among all MPs, the most used ones are Dynamic Movement Primitives (DMPs) [125,126], which characterize a movement by means of a second-order dynamical system, using a position error, a velocity term and an excitation function for obtaining the acceleration profile generating the movement:

$$\begin{gathered} \ddot{\mathbf{y}}/\tau^2 = \alpha_z \left(\beta_z \left(\mathbf{y}_g - \mathbf{y}\right) - \dot{\mathbf{y}}/\tau\right) + \mathbf{f}(x) \\ \mathbf{f}(x) = \boldsymbol{\Psi}_t^T \boldsymbol{\omega}, \end{gathered} \tag{5.14}$$

where $\mathbf{y}$ is the joint position vector, $\mathbf{y}_g$ the goal/ending joint position, τ a time constant, and x is a transformation of time verifying $\dot{x} = -\alpha_x x/\tau$. In addition, $\boldsymbol{\omega}$ is the parameter vector of size dN_f, N_f being the number of Gaussian kernels used for each of the d-DoF. The parameters $\boldsymbol{\omega}$ fitting the robot behavior from an initial move are usually obtained through least-squares minimization, and then multiplied by a Gaussian weights matrix[2] $\boldsymbol{\Psi}_t = \mathbf{I}_d \otimes \mathbf{g}(x_t)$, $\otimes$ being the Kronecker product, and $\mathbf{g}(x)$ defined as[3]:

$$g_i(x) = \frac{\phi_i(x)}{\sum_j \phi_j(x)} x, i = 1..N_f, \tag{5.15}$$

where $\phi_i(x) = \exp\left(-0.5(x - c_i)^2/d_i\right)$, and c_i, d_i represent the fixed center and width of the ith Gaussian. Fig. 5.1 shows an example of the normalized kernels $\frac{\phi_i(x)}{\sum_j \phi_j(x)}$ and Fig. 5.2 multiplied by the phase variable x seen in Fig. 5.3, which ensures the values of $\mathbf{f}$ are small when $t \to \tau$ and, therefore, the state $\mathbf{y}$ converges to the goal $\mathbf{y}_g$ at the end of the trajectory.

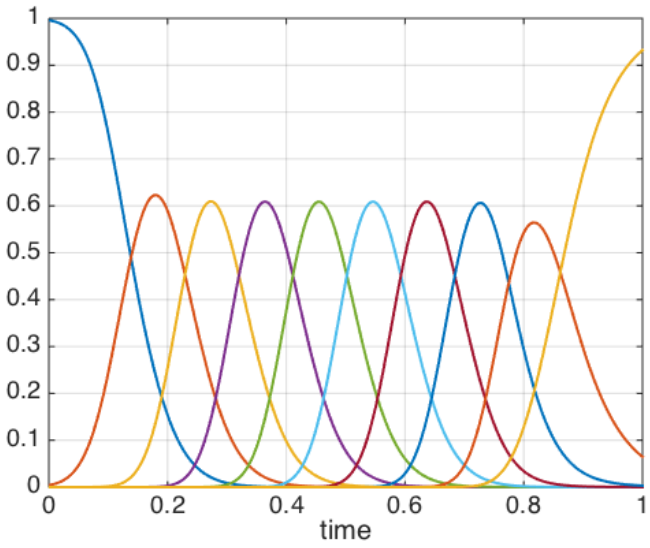


Figure 5.1: An example of $N_f = 10$ basis functions $\phi_i / \sum_j \phi_j$, $j = 1..N_f$.

With this motion representation, the robot can be kinesthetically taught a demonstration movement, to obtain the weights and Gaussians of the motion by using least squares techniques, with the isolated values of $\mathbf{f}$ from Eq. (5.14).

The DMP representation of trajectories has good scaling properties wrt. trajectory time and

[2] In standard DMPs, such a set of radial basis functions represented with Gaussian curves are uniformly distributed in the time domain for each DoF of the trajectory. The weights corresponing to each of these Gaussians are then obtained through least squares techniques. This approach has been used throughout this thesis.

[3] Note that other characterizations, such as $g_i(x) = \frac{\phi_i(x)}{\sum_j \phi_j(x)} x(\mathbf{y}_g - \mathbf{y}_0)$, allow for a better goal rescaling [125].

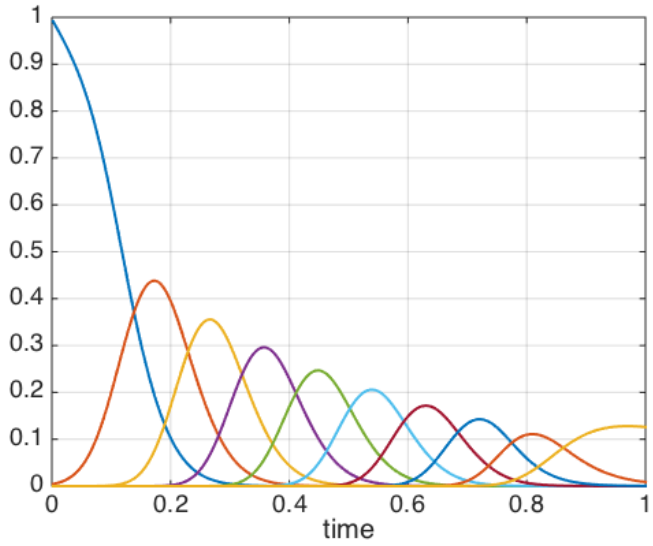


Figure 5.2: An example of $N_f = 10$ basis functions $x \cdot \phi_i / \sum_j \phi_j$, $j = 1..N_f$.

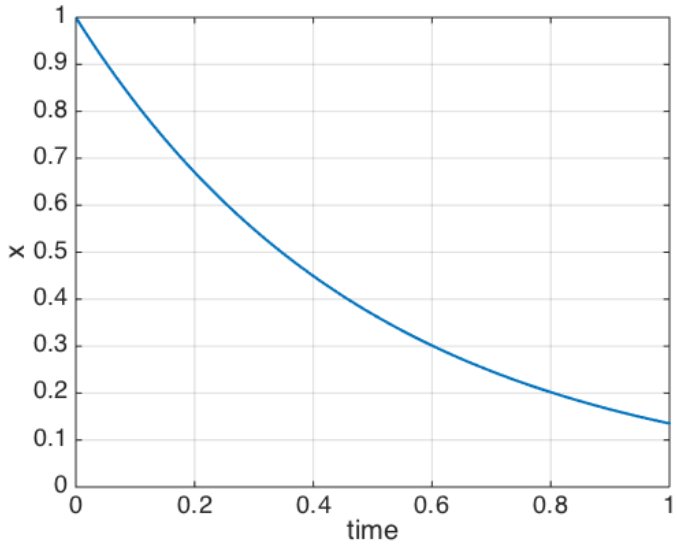


Figure 5.3: Plot of the phase variable x wrt. time.

initial/ending positions, has an intuitive behavior, does not have an explicit time dependence and is linear in the parameters, among other advantages [125]. For these reasons, DMPs are being widely used with Policy Search (PS) RL [12, 118], where the problem of finding the best policy (i.e.: MP parameters) becomes a case of stochastic optimization. Given an initial trajectory demonstrated to the robot, we can obtain the DMP parameters representing it as $\boldsymbol{\mu}_\omega$. However, PS algorithms also require an exploration magnitude for generating new samples, i.e.: a parameter variance $\boldsymbol{\Sigma}_\omega$. In PS with DMPs, such variance needs to be arbitrarily defined by the user, or obtained through several demonstrations. Next, we present an alternative designed to probabilistically fit a set of demonstrations.

5.2.2 Probabilistic movement primitives

Probabilistic Movement Primitives (ProMP) [127] are a general approach to learn and encode a set of similar motion trajectories that present time-dependent variances over time as seen in Fig. 5.4. Given a number of basis functions per DoF N_f as those in Fig. 5.1, ProMP use an $N_f \times 2$ time-dependent matrix $\boldsymbol{\Phi}_t = [\phi_t, \dot{\phi}_t]$ to encode position and velocity, ϕ_t being the vector of normalized kernel basis functions (e.g., uniformly distributed Gaussian basis function over time), thus the position and velocity state vector $\mathbf{y}_t$ can be represented as

$$\mathbf{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t^T \boldsymbol{\omega} + \boldsymbol{\epsilon}_y, \tag{5.16}$$

where $\boldsymbol{\epsilon}_y \sim \mathcal{N}(0, \boldsymbol{\Sigma}_y)$ is a zero-mean Gaussian noise representing fitting error and system noise and the weights $\boldsymbol{\omega}$ are also treated as random variables with a distribution

$$p(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega). \tag{5.17}$$

This distribution can be fitted, given a set of demonstration trajectories $\boldsymbol{\tau}_k = \{\mathbf{y}_t^k\}_{t=1..N_t}$, $k = 1..N_k$, by getting the weights $\boldsymbol{\omega}_k$ of each demonstration with least squares techniques. Subsequently, the parameters of the distribution $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Sigma}_y\}$, $\boldsymbol{\Sigma}_y$ being the state covariance, are fitted by a maximum likelihood estimate, i.e., we compute the sample mean and the sample covariance [128] of $\{\boldsymbol{\omega}\}_{k=1..N_k}$. Then the probability of observing a trajectory $\boldsymbol{\tau}$ can be expressed as the product of all timestep probabilities $p(\mathbf{y}_t; \boldsymbol{\theta})$

$$p(\boldsymbol{\tau}; \boldsymbol{\theta}) = \prod_t p(\mathbf{y}_t; \boldsymbol{\theta}) = \prod_t \int \mathcal{N}(\mathbf{y}_t|\boldsymbol{\Phi}_t^T \boldsymbol{\omega}, \boldsymbol{\Sigma}_y)\mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega) d\boldsymbol{\omega} \tag{5.18}$$

Due to the probabilistic representation, the ProMP approach can represent motion variability

while keeping other MP properties such as rescalation and linear representation w.r.t. parameters. It also allows for other operations such as modulation via probabilistic conditioning and combination by product [127]. In Fig. 5.4, we show the average and standard deviation of a ProMP and different sample trajectories from its distribution.

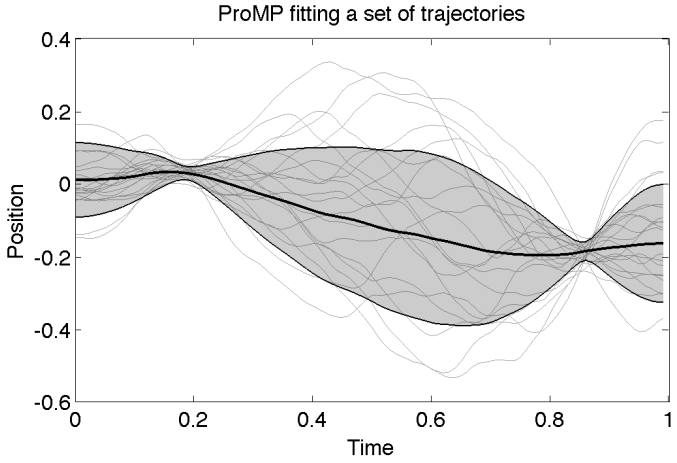


Figure 5.4: ProMP fitting a set of trajectories, the mean and standard deviation for each timestep are shown.

Contrary to the DMP characterization, ProMPs can also be conditioned to impose a viapoint in the trajectory by using probability operations, as well as combining or blending of trajectories.

ProMP control

In order to fully exploit the ProMP, [127] characterize a stochastic controller that reproduces the motion variance. Assuming a known discrete-time linearized dynamics of the system with a time step dt, the robot dynamics equation becomes

$$\mathbf{y}_{t+dt} = (\mathbf{I} + dt\mathbf{A}_t)\mathbf{y}_t + \mathbf{B}_t dt\mathbf{u} + \mathbf{c}_t dt, \tag{5.19}$$

where $\mathbf{A_t}$, $\mathbf{B_t}$, and $\mathbf{c_t}$ are the system, input and drift terms of the first-order Taylor expansion of the dynamical system of the robot.

By using the control signal $\mathbf{u}$ defined as:

$$\mathbf{u} = \mathbf{K}_t\mathbf{y}_t + \boldsymbol{\kappa}_t + \boldsymbol{\epsilon}_u, \tag{5.20}$$

with $\boldsymbol{\epsilon}_u \sim \mathcal{N}(0, \boldsymbol{\Sigma}_u/dt)$ as in [127]. $\mathbf{K}_t$ is a linear gain and $\boldsymbol{\kappa}_t$ a drift term. Inserting (5.20) into

(5.19) results in

$$\begin{aligned} \mathbf{y}_{t+dt} = & \left[(\mathbf{I} + dt\mathbf{A}_t) + dt\mathbf{B}_t\mathbf{K}_t\right]\mathbf{y}_t + \\ & \mathbf{B}_t(\mathbf{u} + \boldsymbol{\epsilon}_u)dt + \mathbf{c}_t dt \\ = & \mathbf{F}\mathbf{y}_t + \mathbf{f} + \mathbf{B}_t dt\boldsymbol{\epsilon}_u, \end{aligned} \tag{5.21}$$

where $\mathbf{F} = \left[(\mathbf{I} + dt\mathbf{A}_t) + dt\mathbf{B}_t\mathbf{K}_t\right]$ and $\mathbf{f} = dt\left(\mathbf{B}_t\mathbf{u}dt + \mathbf{c}_t\right)$. Given the dynamics equation (5.21), the probability of being in state $\mathbf{y}_{t+dt}$ at the next time step is extracted, i.e.,

$$\begin{gathered} p(\mathbf{y}_{t+dt}) = \int \mathcal{N}(\mathbf{y}_{t+dt}|\mathbf{F}\mathbf{y}_t + \mathbf{f}, \boldsymbol{\Sigma}_s dt)\mathcal{N}(\mathbf{y}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)d\mathbf{y}_t = \\ \mathcal{N}(\mathbf{y}_{t+dt}|\mathbf{F}\boldsymbol{\mu}_t + \mathbf{f}, \mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \boldsymbol{\Sigma}_s dt), \end{gathered} \tag{5.22}$$

with $\boldsymbol{\mu}_t = \boldsymbol{\Psi}_t^T\boldsymbol{\mu}_\omega$ and $\boldsymbol{\Sigma}_t = \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}_t$. We can match the control noise matrix $\boldsymbol{\Sigma}_s$ for each timestep by using the cross-correlation between consecutive steps of the trajectory distribution:

$$dt\boldsymbol{\Sigma}_s = \boldsymbol{\Sigma}_{t+dt} - \mathbf{C}_t^T\boldsymbol{\Sigma}_t\mathbf{C}_t,$$

with $\mathbf{C}_t = \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}_{t+dt}$. The controller terms $\mathbf{K}_t$, $\boldsymbol{\kappa}_t$ can be obtained by matching $\{\boldsymbol{\mu}_{t+dt}, \boldsymbol{\Sigma}_{t+dt}\}$ from the system dynamics in Eq. (5.22) and the ProMP model,

$$\mathbf{K}_t = \mathbf{B}^\dagger\left[\left(\dot{\boldsymbol{\psi}}_t^T\boldsymbol{\Sigma}_\omega\dot{\boldsymbol{\psi}}_t - \boldsymbol{\Sigma}_s/2\right) - \mathbf{A}_t\boldsymbol{\Sigma}_t\right]\boldsymbol{\Sigma}_t^{-1},$$

and

$$\boldsymbol{\kappa}_t = \mathbf{B}^\dagger\left[\left(\dot{\boldsymbol{\psi}}_t^T\boldsymbol{\mu}_\omega - \mathbf{A}_t + \mathbf{B}_t\mathbf{K}_t\right)\boldsymbol{\psi}_t^T\boldsymbol{\mu}_\omega - \mathbf{c}_t\right],$$

and the controller noise covariance estimation is given by

$$\boldsymbol{\Sigma}_u = \mathbf{B}^\dagger\boldsymbol{\Sigma}_s\mathbf{B}^{T\dagger}. \tag{5.23}$$

This linearized controller tracks the ProMP as a probability distribution (see [127] for more details). Therefore, it will present larger error-compensating gains when the timestep variability is small and smaller gains when it is large. However, it does need a linear model of the system, as well as a well-conditioned covariance matrix $\boldsymbol{\Sigma}_\omega$, which inverse matrix is crucial to the controller computation.

5.3 Summary

In this chapter, we introduced the basic concepts relative to PS applied to robot motion learning, and suitable motion characterizations used for PS. While DMPs have been widely used throughout the last decade, alternatives like ProMPs are gaining popularity in some applications. We introduced two PS algorithms: PI2 and REPS. While both are state-of-the-art algorithms, PI2 needs the user to set the temperature parameter λ. The choice of such parameter is crucial for learning, and the user needs a prior knowledge of the algorithm's behavior in order to correctly implement it. REPS automatically finds such temperature, denoted η in Eq. (5.7), and the KL divergence bound needed, ϵ, is often set to values between 0.5 and 1 with success. However, the order chosen for the KL constraint in REPS might result in suboptimal solutions in certain problems. In the following chapter, we generalize the REPS algorithm for a better learning behavior in multi-modal problems and, in Chapter 7, we apply Dimensionality Reduction (DR) techniques to both DMPs and ProMPs, with the particularity of making such DR reward-oriented. That allows to encode data in a way we keep the maximum information of the task possible, but also giving much more importance to the higher-reward data when encoding motion.

6

Dual REPS: A Generalization of Relative Entropy Policy Search Exploiting Bad Experiences

Policy Search (PS) algorithms are nowadays widely used for their simplicity and effectiveness in finding solutions for robotic problems. However, most current PS algorithms derive policies by statistically fitting the data from the best experiments only. This means that those experiments yielding a poor performance are usually discarded or given too little influence on the policy update. In this chapter, we propose a generalization of the Relative Entropy Policy Search (REPS) algorithm that takes bad experiences into consideration when computing a policy. The proposed approach, named Dual REPS (DREPS) [129], following the philosophical interpretation of the duality between good and bad, finds clusters of experimental data yielding a poor behavior and adds them to the optimization problem as a repulsive constraint. Thus, considering there is a duality between good and bad data samples, both are taken into account in the stochastic search for a policy. Additionally, a cluster with the best samples may be included as an attractor to enforce faster convergence to a single optimal solution in multi-modal problems.

REPS, which has been introduced in Section 5.1.1, uses a Kullbach Leibler (KL) divergence term for limiting the difference between the former and the new policy in a PS update. However, it has been shown [123] that the ordering of the KL arguments used in REPS - $\mathrm{KL}(\pi(\boldsymbol{\theta})\|q(\boldsymbol{\theta}))$ instead of $\mathrm{KL}(q(\boldsymbol{\theta})\|\pi(\boldsymbol{\theta}))$ - has an averaging-between-solutions behavior, sometimes finding non-optimal solutions due to competition between two or more close local optima. Such KL optimization generates a solution that yields a high probability where data presents a high reward, therefore averaging between modes in Gaussian distributions. Using the reverse ordering would help to find a single solution faster, as such approach would find a solution with low probability values where data has low reward [123], therefore focusing on only one mode. However, there is no closed REPS solution using $\mathrm{KL}(q\|\pi)$ instead of $\mathrm{KL}(\pi\|q)$. While the latter usage of KL seems to be more appropriate for most RL applications, its analytic unsolvability makes its

application impractical. Along this chapter we will show that our proposed DREPS algorithm avoids the aforementioned averaging-between-solutions behavior, exhibiting a good ability to escape from getting trapped in between local maxima. The results in 6.1.3 show that, while having a very similar performance in unimodal learning cases, DREPS outperforms the standard REPS in multimodal problems as the real-robot example presented.

In order to allow our proposed DREPS algorithm to fit in the current trends of RL, we assume that policies are encoded as multivariate normal random distributions. Such encoding, as well as representing the clustered samples by Gaussian distributions, allows us to solve the resulting equations analytically and obtain a closed-form solution.

In the following section, we will detail how to build a clustered data structure for DREPS, followed by the algorithm's derivation, which is done similarly to REPS and other information-theoretic Policy Search approaches [130].

6.1 The Dual REPS Algorithm

The idea behind Dual REPS (DREPS) is to use clustered badly-performing data samples as a repulsive field and good-performing ones as an attractor within the policy search algorithm. For this purpose, we assume that we can cluster the data from a set of experiments and encode such information as a constraint in the optimization problem. The natural way of adding this new constraint has been to set a minimum/maximum KL-divergence between the bad/good data clusters and the new policy. This will act as a repulsive/attractive field from these clusters in the policy update. This results in a dichotomic effect, limiting the search space when finding an optimal solution, given the samples available, for the policy update. The optimization problem presented in Eq. (6.2) differs from the REPS optimization (see Eq. 5.6) in that, while still maximizing the gain in expected reward for the updated policy, two additional restrictions are added. These two restrictions add a minimum required distance from the new policy to a number of clusters of badly-performing samples, as well as a bound on the KL divergence wrt. the best clusters of data. The first restriction (repulsive) prevents the new policy to stay in suboptimal areas of the optimization space, instead of ignoring low-performing samples by giving them zero importance (as happens in REPS). The second restriction then allows for a greedier behaviour of the policy search when more than one optimum exists, helping to converge faster when possible (see video in Appendix B.5).

In this section, we will first explain how to select and fit these data clusters, which will be labeled *bad/good data*, to later present the whole mathematical derivation of the proposed DREPS algorithm. The clusterization presented in this chapter is the result of a trial-and-error attempt at finding a reliable way of clustering parameter data with aggregated rewards which

could be used in the DREPS algorithm. While such proposed clustering has proved effective, we note that it is just a tool to obtain the input needed for the DREPS algorithm itself. The clusters are fitted with Gaussian distributions that are then used for the policy search update.

Throughout this section, we will use C_{low} as the set of low-performing data clusters, with its cardinal represented as $|C_{low}|$, and C_{high} as the set of $|C_{high}|$ high-performing data clusters. Additionally, for simplicity of notation we will use $C = C_{low} \cup C_{high}$ with cardinal $c = |C|$. It is important to note that not all the samples obtained are necessarily classified in one of these clusters, and there can also be an overlaping of clusters. This is a design decision to give more freedom to the clustering algorithm defined in the following section.

6.1.1 Clustering

The low-performing and high-performing samples may be spread out in the parameter space. Therefore, we have opted for grouping the samples with a bad/good reward in several clusters, so as to represent them by Gaussians to be included in the policy update. In such clustering, we assume the reward function is smooth almost everywhere from the mathematical perspective, as well as a good repeatability of the reward function wrt. policy parameters.

In order to obtain such clusters, we decided to use *K-means* clustering, considering both the sample vectors and the rewards generated by them. Hence, we append a transformed reward $f(r_k)$, with $r_k \triangleq \mathbf{R}(\boldsymbol{\theta}_k)$, to every sample $\boldsymbol{\theta}_k \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$, $k = 1, ..., N$, N being the number of rollouts per policy update. Then, we use these vectors $\left[\boldsymbol{\theta}_k^T - \boldsymbol{\mu}_\omega^T, f(r_k)\right]^T$ as input to a *K-means* clustering with a given fixed number of clusters. The usage of such $f(r_k)$ is to properly scale the relative importance between rewards and parameters in the clustering. Otherwise, either one or the other could have a too-large influence, as seen in Fig. 6.1. We initially tested with the *K-means* clustering algorithm in MATLAB. We used the *K-means* clustering algorithm in MATLAB. However, such implementation [131] initializes the cluster centers by randomly selecting a number of points from the dataset. This resulted in a non-deterministic way of clustering points, so the clusters are initialized using the algorithm proposed in [132] instead.

We perform two independent clustering processes on the same data, using two different reward transformations, to obtain the low- and high-performing clusters. The reason for these two transformations is to increase the importance of the reward dimension in the clustering (amplifying either the low or high values), while maintaining the proportionality in the samples. Note that, as we are clustering the data twice with the last component being different, some clusters in C_{high} might overlap with those in C_{low}.

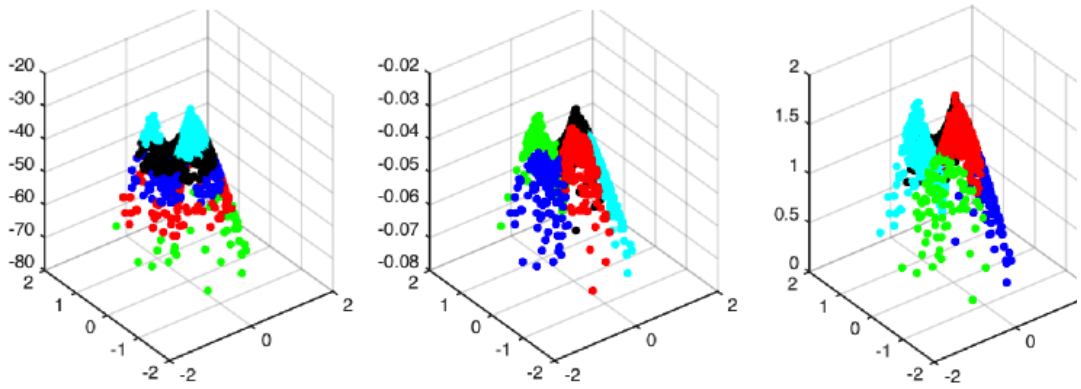


Figure 6.1: Result of applying *K-means* clustering to the 3-dimensional data points consisting of two random variables and the transformed reward without the rescaling in Eq. (6.1). Left column shows the results with too much relative importance on rewards, disregarding parameter variability. On the contrary, the center column shows the clustering results with too small influence of the rewards and right column shows the proposed relative weighting.

Obtaining the high-performing clusters

Given the vector of rewards $\mathbf{r} = \{r_1, ..., r_N\}$, we define:

$$f(r_k) = \rho \cdot \sqrt{\text{tr}(\mathbf{\Sigma}_\omega)} \frac{r_k - \min(\mathbf{r})}{\max(\mathbf{r}) - \min(\mathbf{r}) + 10^{-9}}, \tag{6.1}$$

where ρ is a relative importance weight, which will keep the sample reward importance proportional to the sample variability during the clustering part of DREPS. This transformation of the reward firstly normalizes the values to $[0, 1]$ and then scales such values to a similar magnitude to that of the parameter variance. Otherwise, as mentioned earlier, undesired clusterization like the one seen in Fig. 6.1 could be obtained, where data is clustered by reward value only. A value of $\rho = 10/D$, D being the number of parameters, has been used throughout this chapter. Once the data has been prepared, we run the *K-means* clustering algorithm and obtain a cluster label for each sample, indicating to which cluster it has been assigned. Taking the average transformed reward for each of the clusters, we separate them into two groups (using a $1-$dimensional *K-means* clustering with 2 clusters). The group with the best average rewards will be the clusters we will consider as *high-performing clusters* and we will gather them in the set C_{high}. We may define a maximum number of low-performing clusters, consider only a single cluster, or let the algorithm choose the number of high-performing clusters.

Algorithm 6.1: *K-means* clustering for DREPS

Input:
Sample vector $\boldsymbol{\theta}_k$, rewards r_k, $\forall k = 1, ..., N$
Number of dual clusters c

1: Transform the rewards to more discriminating values $f(r_k)$ with (6.1).
2: Perform standard *K-means* clustering with $[\boldsymbol{\theta}_k, f(r_k)]$ and obtain c clusters.
3: Compute the average transformed reward $\overline{f(r)}_i$ for each cluster $i = 1..c$.
4: Choose the clusters with the best average transformed reward, manually or with a 2-cluster *K-means* approach, and assign them to C_{high}.
5: Transform the Rewards by using $1/r_k$ instead of r_k in (6.1) to obtain $f(1/r_k)$.
6: Perform standard K-means clustering with $[\boldsymbol{\theta}_k, f(1/r_k)]$ and obtain c clusters.
7: Choose the clusters with the highest average inverse reward, manually or with a 2-cluster *K-means* approach, and assign them to C_{low}.
8: **for** $i \in C$ **do**
9: Compute μ_i, Σ_i with reward-Weighted Maximum Likelihood Expectation (WMLE) using points assigned to cluster i, using the transformed rewards as weights.
10: **end for**

Obtaining the low-performing clusters

The only difference with the just described clustering process is that here we use $1/\max(r_k, 10^{-9})$ instead of r_k in (6.1). In this way, we obtain a set of clusters C_{low} with low-performing data.

Next, we fit each cluster in C with a Normal distribution $g_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ using their associated transformed rewards as weights in an WMLE to obtain the resulting parametrizations for each cluster $i \in C$, $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$. Given the reward function in Fig. 6.2, Fig. 6.3 shows an example of the classification of the sample points with *K-means* clustering, while Fig. 6.4 displays their associated rewards, and the Normal distributions resulting from the WMLE using a total of 3 clusters, 2 of them being considered as having a low performance. No high-performing cluster is used as attractor in this example. The plot shows the effectiveness of the clustering algorithm at detecting poorly performing areas on the policy space. In Algorithm 6.1, we summarize the process of computing the clusters given the data samples.

6.1.2 DREPS derivation

Given the information provided by the clustering in the previous section, we will use the computed clusters, represented as Gaussian distributions $g_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, $i \in C$, as repulsive or

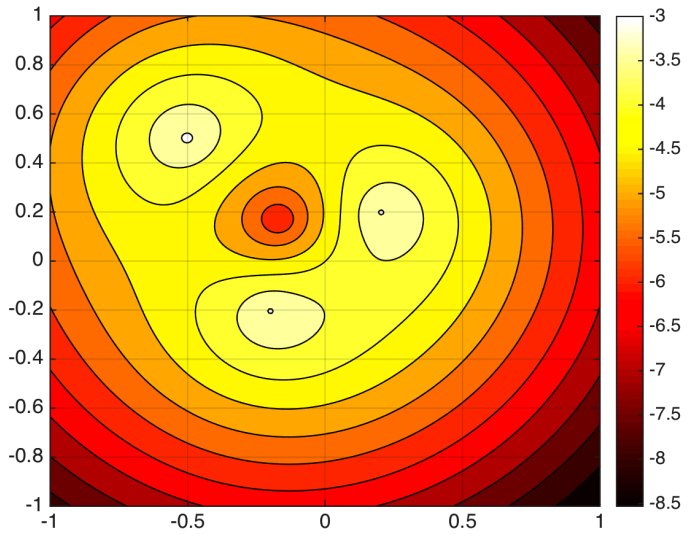


Figure 6.2: Reward function used as a clustering illustrative example and in the experimental section (see (6.16)).

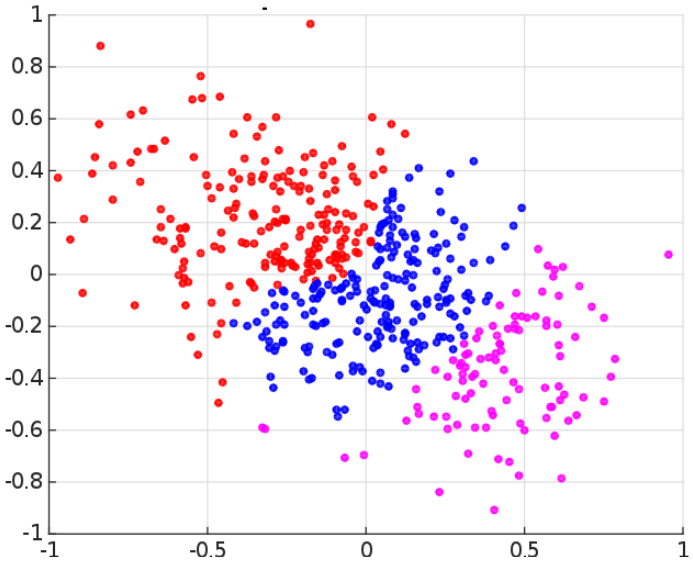


Figure 6.3: Result of classification with *K-means* clustering of the 3-dimensional data points consisting of two random variables and the transformed reward for the reward function in Fig. 6.2.

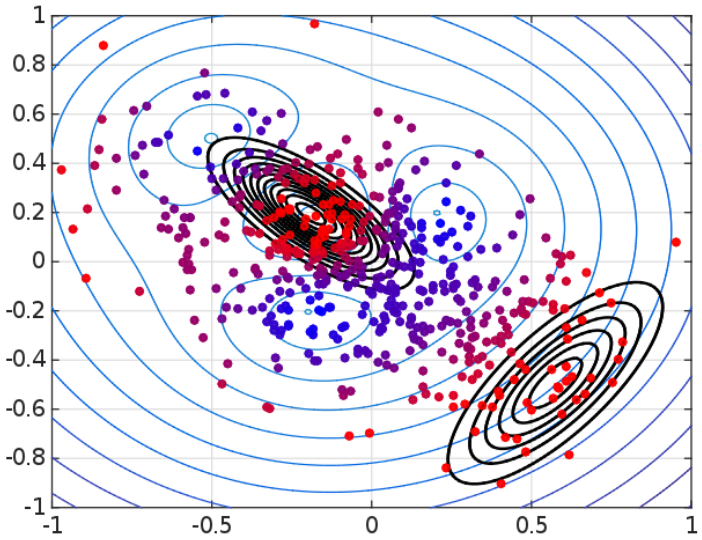


Figure 6.4: Result of applying Algorithm 6.1 to the data points color coded from red (low reward) to blue (high reward). After clustering using the transformed reward as displayed in Fig. 6.3, the two lowest-performing clusters are fitted with Gaussians using WMLE and are here shown in black.

attractive data for the optimization problem, which now becomes:

$$\begin{aligned} \pi^* = \text{argmax}_\pi \int \pi(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta})d\boldsymbol{\theta} \\ \text{s.t. } \epsilon \geq \text{KL}(\pi\|q) \\ 1 = \int \pi(\boldsymbol{\theta})d\boldsymbol{\theta} \\ \xi \leq \text{KL}(\pi\|g_i)\ ,\ i \in C_{low} \\ \text{KL}(\pi\|g_i) \leq \chi\ ,\ i \in C_{high}, \end{aligned} \tag{6.2}$$

where ϵ is the bound on the KL-divergence for the REPS algorithm, and ξ, χ are the minimum and maximum KL-divergence we want to have between the new policy and the precomputed low-performing and high-performing clusters, respectively. Note that the condition $\epsilon \leq \text{KL}(\pi\|q)$ could be included in the C_{high} restriction. However, we decided to keep it separate to make clear that here the KL-divergence is not with respect to a cluster, but with respect to the previous policy parameters and will always be maintained, while C_{high} may be an empty set and represents a more local influence. The solution of (6.2) can be found analytically by using Lagrange multipliers and has the form

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})^{\frac{\eta}{\eta+\omega-\nu}} \prod_i g_i(\boldsymbol{\theta})^{\frac{\omega_i-\nu_i}{\eta+\omega-\nu}} \exp\left(\frac{\mathbf{R}(\boldsymbol{\theta})}{\eta+\omega-\nu}\right), \tag{6.3}$$

where η is the Lagrange multiplier of the first KL constraint in (6.2) and $\nu = \sum \nu_i$, $\omega = \sum \omega_i$ are the multipliers for the other KL constraints. These variables can be found by minimizing the dual function of the optimization problem (derived in eqs. (6.5)-(6.12)). Note that in this chapter we will use ν and $\boldsymbol{\nu} = \left[\nu_1, ..., \nu_{|C_{low}|}\right]$. Analogously, we will be using ω and $\boldsymbol{\omega}$. Hence, the optimal Lagrange multipliers are the ones obtained by solving:

$$\{\eta^*, \boldsymbol{\nu}^*, \boldsymbol{\omega}^*\} = \text{argmin}_{\eta,\boldsymbol{\nu},\boldsymbol{\omega}} h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}), \tag{6.4}$$

Given the optimization problem (6.2), we can compute the lagrangian as:

$$\begin{aligned} \mathcal{L} = \int \pi(\boldsymbol{\theta})\mathbf{R}(\boldsymbol{\theta})d\boldsymbol{\theta} + \eta\left(\epsilon - \int \pi(\boldsymbol{\theta})\log\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}d\boldsymbol{\theta}\right) \\ + \sum_{i\in C_{low}} \nu_i\left(\int \pi(\boldsymbol{\theta})\log\frac{\pi(\boldsymbol{\theta})}{g_i(\boldsymbol{\theta})}d\boldsymbol{\theta} - \xi\right) \\ \sum_{i\in C_{high}} \omega_i\left(\chi - \int \pi(\boldsymbol{\theta})\log\frac{\pi(\boldsymbol{\theta})}{g_i(\boldsymbol{\theta})}d\boldsymbol{\theta}\right) + \lambda\left(\int \pi(\boldsymbol{\theta})d\boldsymbol{\theta} - 1\right) \end{aligned} \tag{6.5}$$

where η, ν_i,ω_i, $\forall i$ are positive. Differentiating with respect to $\pi(\boldsymbol{\theta})$ (and omitting $\boldsymbol{\theta}$ for simplicity) we obtain

$$\frac{\partial\mathcal{L}}{\partial\pi} = \mathbf{R} - \eta(\log\pi - \log q + 1) + \lambda + \sum_{i\in C_{low}} \nu_i(\log\pi - \log g_i + 1) - \sum_{i\in C_{high}} \omega_i(\log\pi - \log g_i + 1) \tag{6.6}$$

which, setting $\frac{\partial\mathcal{L}}{\partial\pi} = 0$ and isolating $\log\pi$ becomes:

$$\log\pi = \frac{\mathbf{R}}{\eta + \omega - \nu} + \frac{\eta\log q}{\eta + \omega - \nu} + \frac{\sum_{i\in C_{high}} \omega_i \log g_i}{\eta + \omega - \nu} - \frac{\sum_{i\in C_{low}} \nu_i \log g_i}{\eta - \nu} - \frac{\eta + \omega + \lambda - \nu}{\eta + \omega - \nu} \tag{6.7}$$

with $\nu = \sum_{i\in C} \nu_i$, and setting $Z = \exp\left(\frac{\eta+\omega+\lambda-\nu}{\eta+\omega-\nu}\right)$, we obtain

$$\pi = Z^{-1} q^{\eta/(\eta+\omega-\nu)} \prod_{i\in C} g_i^{\omega_i - \nu_i/(\eta+\omega-\nu)} \exp\left(\frac{\mathbf{R}}{\eta + \omega - \nu}\right) \tag{6.8}$$

where, given that $1 = \int \pi(\boldsymbol{\theta})d\boldsymbol{\theta}$,

$$Z = \int_{\boldsymbol{\theta}} q^{\eta/(\eta+\omega-\nu)} \prod_{i\in C} g_i^{\omega_i - \nu_i/(\eta+\omega-\nu)} \exp\left(\frac{\mathbf{R}}{\eta + \omega - \nu}\right) d\boldsymbol{\theta}. \tag{6.9}$$

Now, reinserting (6.8) into (6.5), we obtain a dual function for the lagrangian problem:

$$h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) = \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) + \lambda + \eta + \omega - \nu \tag{6.10}$$

where, isolating $\lambda + \eta + \omega - \nu$ from Z in equation (6.9) and inserting it into (6.10):

$$\begin{aligned} h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) &= \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) \\ &+ (\eta + \omega - \nu) \log \textstyle\int_{\boldsymbol{\theta}} q^{\eta/(\eta+\omega-\nu)} \prod_{i \in C} g_i^{\omega_i - \nu_i/(\eta+\omega-\nu)} \exp\left(\frac{\mathbf{R}}{\eta+\omega-\nu}\right). \end{aligned} \tag{6.11}$$

We can now replace the integral over a sum of samples to obtain the dual objective function:

$$\begin{aligned} h(\eta, \boldsymbol{\nu}, \boldsymbol{\omega}) &= \eta\epsilon + \sum_{i \in C_{high}} (\omega_i \chi) - \sum_{i \in C_{low}} (\nu_i \xi) + \\ &+ (\eta + \omega - \nu) \log \left[\frac{1}{N} \sum_{k=1}^{N} q_{(k)}^{\frac{\nu-\omega}{\eta+\omega-\nu}} \prod_{i \in C} g_{i\,(k)}^{\frac{\omega_i - \nu_i}{\eta+\omega-\nu}} \exp\left(\frac{r_k}{\eta + \omega - \nu}\right) \right]. \end{aligned} \tag{6.12}$$

This dual function can be evaluated provided we can compute the probability of a given trajectory for both the previous policy q and the dual policies g_i. These can be computed from the direct policy evaluation or, in cases where the outcome is a sequence of states, by multiplying the transition probabilities for all the timesteps of a sequence. For numerical stability reasons, we recommend to directly compute the log-probability of such normal distribution.

From the mathematical perspective, there is no guarantee that this problem will always be convex for ν, thus in order to minimize the dual function h, we set a minimum value for ν_i in the active-set optimization of the dual function, it being an indicator of the minimum influence we want the dual policies to have. If g_i are defined by fitting a normal distribution given some clustered samples with their associated rewards (assuming rewards are negative, and closer to zero is considered better), we can, for example, set $\nu_i = \nu \frac{1}{\sqrt{|\overline{R}_i|}}$, with $\nu = \sum_i \nu_i$ and $\overline{R}_i$ the average reward for the i-th cluster.

Additionally, in some circumstances the solution provided by the solver might not be fully respecting the ϵ bound on the KL-divergence. This comes from trying to find a probability distribution with a min/max dissimilarity with respect to other distributions, which could then become a set of restrictions impossible to comply with. For that reason, the KL-divergence of the solution found was evaluated after the policy update, and in case $KL(\pi \| q) > \epsilon$, the gradient of the KL of the solutions found with respect to ν_i, χ and ξ was iteratively obtained, performing gradient descent on these parameters until a suitable solution within the KL-divergence bound was found. In order to perform such gradient descend, it is vital that the *K-means* clustering

initialization is performed in a deterministic manner, as in [132]. Furthermore, when no convergence is reached after a certain number of iterations, $\boldsymbol{\nu}$ is set to zero for that policy update and the optimization is performed only with attractor Gaussians.

Thus, once the Lagrange multipliers $\eta, \boldsymbol{\nu}, \boldsymbol{\omega}$ have been found, one can update the policy by using WMLE, with the weights d_k for each rollout coming from the samples and the solution form shown in (6.3):

$$d_k = q_{(k)}^{\frac{\nu-\omega}{\eta+\omega-\nu}} \prod_{i \in C} g_{i\,(k)}^{\frac{\omega_i-\nu_i}{\eta+\omega-\nu}} \exp\left(\frac{r_k}{\eta+\omega-\nu}\right). \tag{6.13}$$

In Algorithm 6.2 we summarize the DREPS algorithm for clarity.

Algorithm 6.2: Dual Relative Entropy Policy Search (DREPS)

Input:
Parameters ϵ, ξ, χ, and rollouts per update N
Previous policy $q(\boldsymbol{\theta})$

1: **for** $k = 1..N$ **do**
2: Perform an experiment using $\boldsymbol{\theta}_k$, a sample from the policy $q(\boldsymbol{\theta})$. Compute reward r_k.
3: **end for**
4: Perform both steps of *K-means* clustering as defined in Section 6.1.1 and obtain $g_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, for $i \in C$
5: Compute the probabilities of each rollout for the previous policy $q(\boldsymbol{\theta}_k)$ and the dual policies $g_{i\,(k)} = g_i(\boldsymbol{\theta}_k)$ for $k = 1..N$ and $i \in C$.
6: Perform optimization to find $\eta, \boldsymbol{\nu}, \boldsymbol{\omega}$ with the dual function in (6.4).
7: Find weights d_k for each rollout k as in (6.13).
8: Perform WMLE with the obtained weights d_k and parameter vectors $\boldsymbol{\theta}_k$ to find the new policy π.

Note that, for $\boldsymbol{\nu} = 0$ and $\boldsymbol{\omega} = 0$, the effect of the clustered data would be none and the algorithm should behave exactly as REPS. Indeed, for $\boldsymbol{\nu} = 0$ and $\boldsymbol{\omega} = 0$ the solution in (6.3) becomes:

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\left(\frac{\mathbf{R}(\boldsymbol{\theta})}{\eta}\right), \tag{6.14}$$

and the dual function to optimize is

$$h(\eta) = \eta\epsilon + \eta \log\left[\frac{1}{N}\sum_{k=1}^{N} \exp\left(\frac{r_k}{\eta}\right)\right], \tag{6.15}$$

which are the REPS solution and the dual function, respectively. Thus, setting the influence of the dual policies g_i to zero, we can see that our proposed algorithm reduces to REPS and, therefore, it is a generalization of REPS. An experiment with a real robot described at the end of Section 6.1.3 experimentally confirms this theoretical remark in unimodal problems.

6.1.3 Experiments

In this section, we present three experimental setups to assess the performance of our proposed algorithm, especially in multi-modal problems: First, a 2-D example of a multi-modal reward function, and second, a multi-modal real robotic problem. Finally, an unimodal experiment is also shown.

Multi-modal 2D reward function

To evaluate how the proposed algorithm performs, we built an example task in which the policy is to sample points $\boldsymbol{\theta}_k$, $k = 1, ..., N$ in a 2-dimensional space and, for each sample, evaluate a reward function r_k consisting in a high reward at three given points $\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \boldsymbol{\psi}_3$ and very low reward in between:

$$r_k = 10\| \sum_{i=1..3} \left(\frac{\boldsymbol{\psi}_i}{3} \right) - \boldsymbol{\theta}_k\| - 5 \sum_{i=1..3} \|\boldsymbol{\psi}_i - \boldsymbol{\theta}_k\| \tag{6.16}$$

The reward function is displayed in Fig. 6.2, where one can note that there are 3 possible candidates for an optimal solution.

To find the optimal point on the plane, we initialize the policy with $\boldsymbol{\mu}_\omega = \mathbf{0}$ and $\boldsymbol{\Sigma}_\omega = \mathbf{I}$, and 100 samples are evaluated for every policy update, reusing up to 400 previous samples. When using REPS with a KL bound of $\epsilon = 0.5$ for this optimization problem, we noticed that the learning curve had a plateau in most cases (see Fig. 6.5), corresponding to the algorithm averaging the rewards of two of the optimal points (see Fig. 6.6). The REPS algorithm keeps obtaining samples near both candidates and cannot improve the policy further until significantly more samples get closer to one of the candidates than the other, moving the policy towards one of the solutions. As a result, the more rollouts per policy update used, the more likely REPS is to stay longer in such plateau.

If, instead of REPS, we use our proposed approach DREPS, the effect of the repulsive Gaussian in the middle allows the algorithm to quickly avoid this plateau and keep on with the optimization. We compared the performance of a REPS algorithm (REPS), a dual REPS algorithm with 3 repulsive Gaussian, and none attractive (nDREPS), and the full DREPS algorithm with one attractive Gaussian and up to 4 repulsive ones (DREPS). In the latter case, we let the algorithm itself decide how many repulsive clusters it would use with a 2-cluster *K-means*, as explained in Section 6.1.1, i.e.: $|C_{low}| \leq 4$, $|C_{high}| = 1$, and parameters $\xi = 5$, $\chi = 2\epsilon$. We performed 50 learning experiments for both REPS and DREPS, and the results are displayed in Fig. 6.7, where we can see that nDREPS performs better than REPS, but both are outperformed by the full DREPS. A video comparing the evolution of REPS vs. DREPS can be found in Appendix

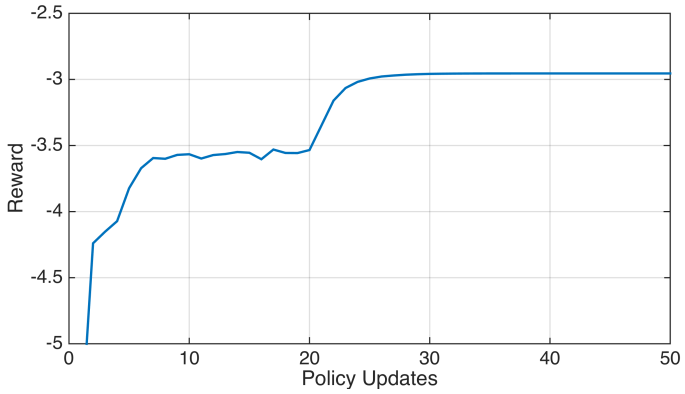


Figure 6.5: Learning curve of the REPS algorithm for the 2D optimization example. The algorithm gets stuck in a plateau averaging between two solutions (see Fig. 6.6) between the 7th and 21th policy updates.

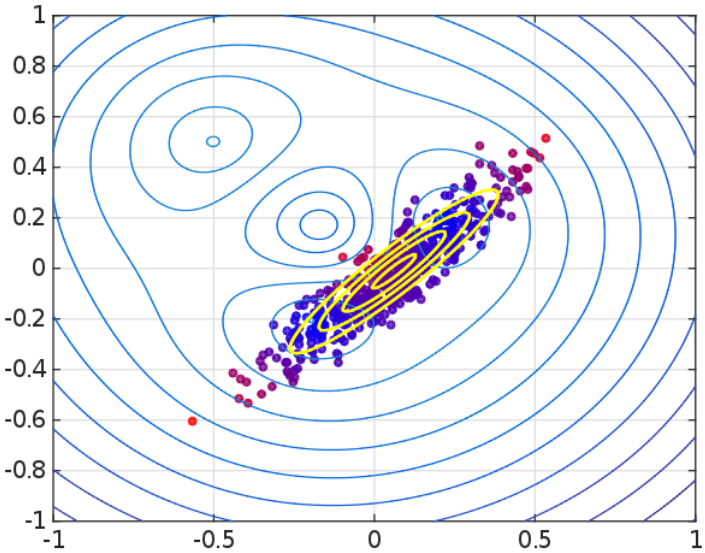


Figure 6.6: Gaussian policy resulting from applying REPS to the 2D optimization example. As shown here, REPS averages between two solutions, and keeps doing so for several iterations, as can be seen in the video included in Appendix B.5.

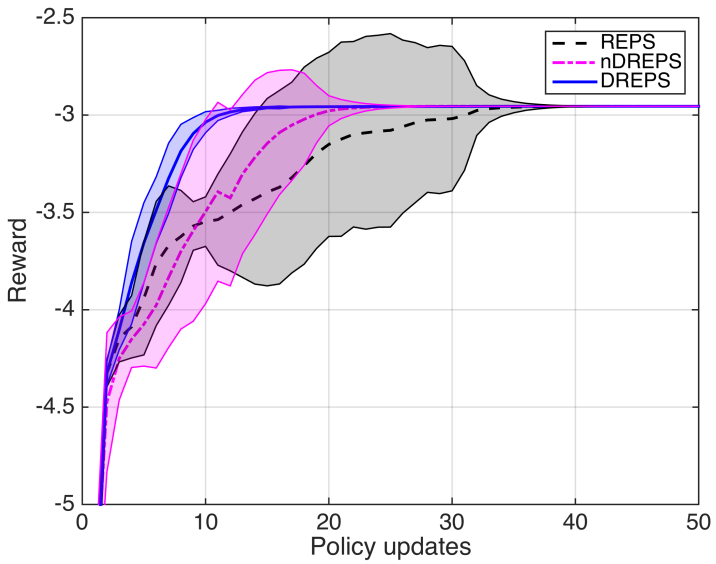


Figure 6.7: The learning curves for the 2D optimization example, averaged for 50 experiments each (mean and 2-standard deviations are plotted), show the advantage of using the DREPS algorithm.

B.5. The scalability of the proposed approach has been assessed using the same problem in a larger-dimensional parameter space, as also seen in the Appendix.

Real robot multi-modal problem

As a second experiment, we programmed a Barrett WAM robot so that its end-effector would follow a straightline trajectory with fixed orientation (facing down) and fixed z component, from a starting position towards a goal position. Two bottles were added on the way as seen in Fig. 6.8 and, using RL, the robot had to adapt the trajectory to an *S-shaped* motion that would not knock down any of the bottles. Within this problem, we learned the task using a compliant controller as defined in Part I of this thesis in three different manners: Through human guidance [22, 23, 133], with REPS [121] and with DREPS.

Regarding the reward function to optimize, an initial approach was taken with strong penalizing terms for knocking down the bottles and the length of the trajectory:

$$\mathbf{R} = -2N_{bottlesdown} - 0.15L_{trajectory}, \tag{6.17}$$

where $N_{bottlesdown}$ is the number of bottles knocked down (0, 1 or 2) and $L_{trajectory}$ is the trajectory length in meters. The relative weights of these terms were set to 2 and 0.15, respectively, giving a higher importance to task accomplishment in the reward function, as usually done in

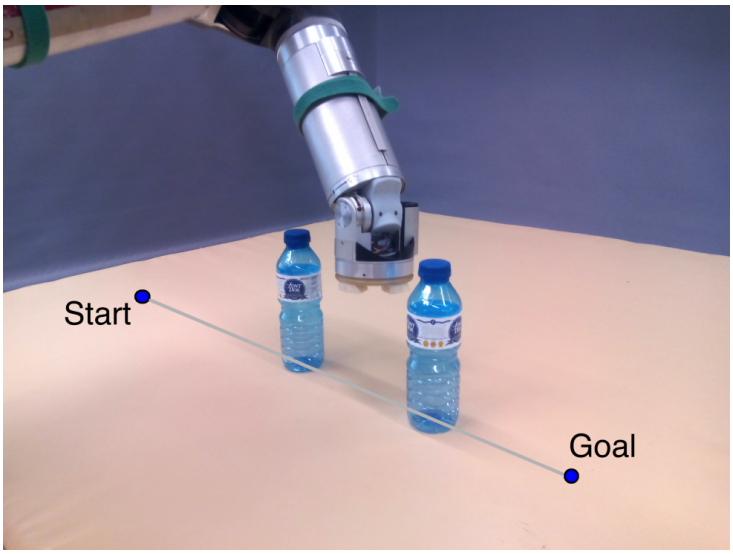


Figure 6.8: Experimental setup for a real robot experiment where the robot must learn to perform an S-shaped motion between the two bottles.

literature [12].

Human guidance

In order to learn a trajectory through visual guidance, in [133] we used a Microsoft Kinect camera that would track a person while performing a straight line movement. At a certain point of the trajectory, the human user would raise his hand and the robot would enter the visual imitation state. The human can then guide the robot's end-effector in-between the bottles and raise his hand again once such part of the trajectory is finished. The robot would then go back to the closest point of the straight line trajectory. Then, the directional points provided at a $10Hz$ rate by the camera are stored, translated from Cartesian poses to joint coordinates in a similar manner as shown in the visual tracking application in Section 4.3.2 (see Fig. 4.10). The human always taught the robot the *same S* motion, leaving always the same first bottle on the left.

After a number of reproductions, the obtained trajectories would be adapted as seen in Fig. 6.9, where the two characteristic points of each blue trajectory (human takes control, human releases control) are shown in red. The common parts - in thick black - of the trajectories are then adapted with a gradient descent time alignment - shown in green - and finally, the mean trajectory shown at the bottom is obtained through a reward-weighted average of the obtained trajectories.

REPS and DREPS

The trajectory was encoded as a Dynamic Movement Primitive (see Section 5.2.1) initialized to a straightline with 10 Gaussians per DoF equally spaced in time, to a total of 20 parame-

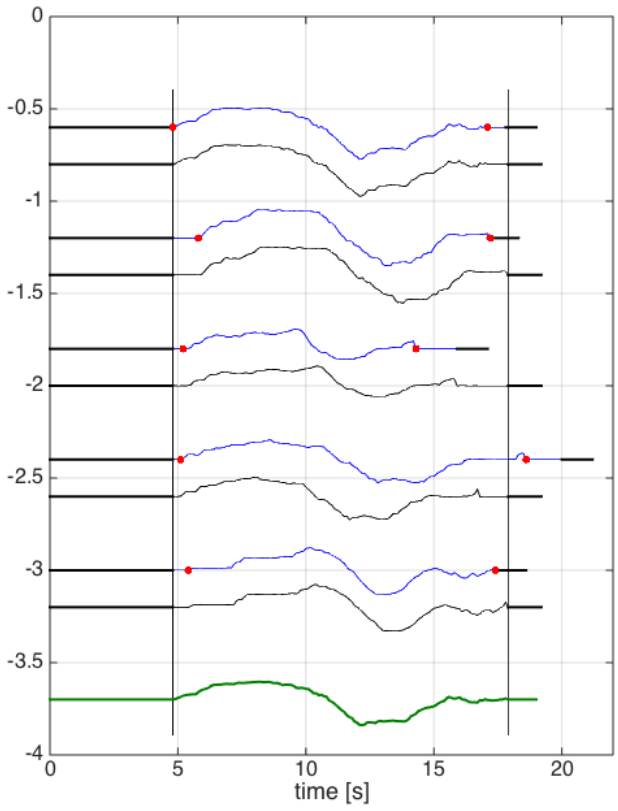


Figure 6.9: Time alignment of several trajectories obtained through guidance.

ters, representing the linear multipliers of such fixed Gaussians, to characterize the x and y components of the trajectory. The same implementation of DREPS as in the previous simulated experiment, with identical algorithmic parameters, was used.

With this setting, DREPS outperformed REPS in convergence velocity, as we can see in Fig. 6.10. Obviously, all the final solutions obtained with both REPS and DREPS were of the two left-most cases of Fig. 6.11, which was to be expected given the reward function in (6.17), but they were not the desired solutions.

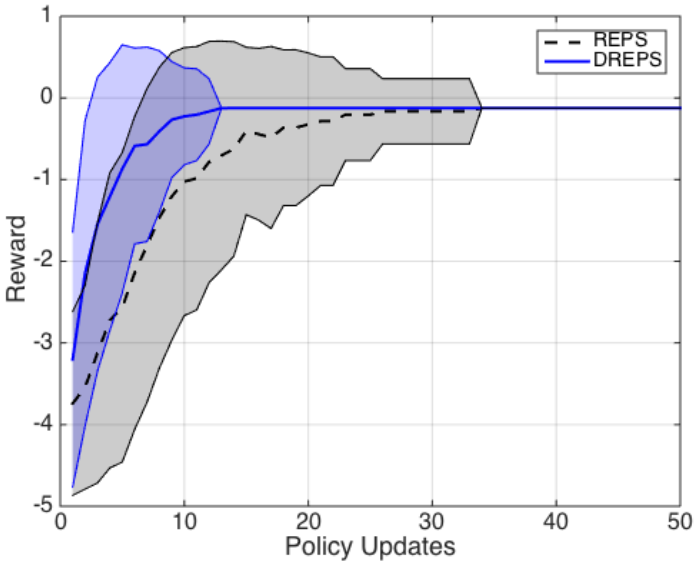


Figure 6.10: Learning curves for REPS and DREPS with the reward function in (6.17). Policy updates were calculated after every 50 rollouts.

For this reason, we added a term penalizing the fact that the robot would not cross between the bottles. To do so, we evaluated in which side of the line drawn in Fig. 6.8 the arm was when passing by each of the two bottles, and a term was added to the reward function penalizing when it was on the same side. Note that this term allows for symmetric solutions as the two right-most ones, plotted in blue, in Fig. 6.11. The new reward function would then be:

$$\mathbf{R} = -2N_{bottlesdown} - 0.15L_{trajectory} - 4\mathbf{I}_{cross}, \tag{6.18}$$

where $\mathbf{I}_{cross}$ indicates whether the robot did or did not cross between the bottles ($\mathbf{I}_{cross} = 1$ in case the robot did not cross, and $\mathbf{I}_{cross} = 0$ in case it crossed). The relative weight of such added term was set to 4 to have the same negative effect in the reward function as if knocking down the two bottles.

We also performed 50 simulated experiments of 100 policy updates with 50 rollouts each

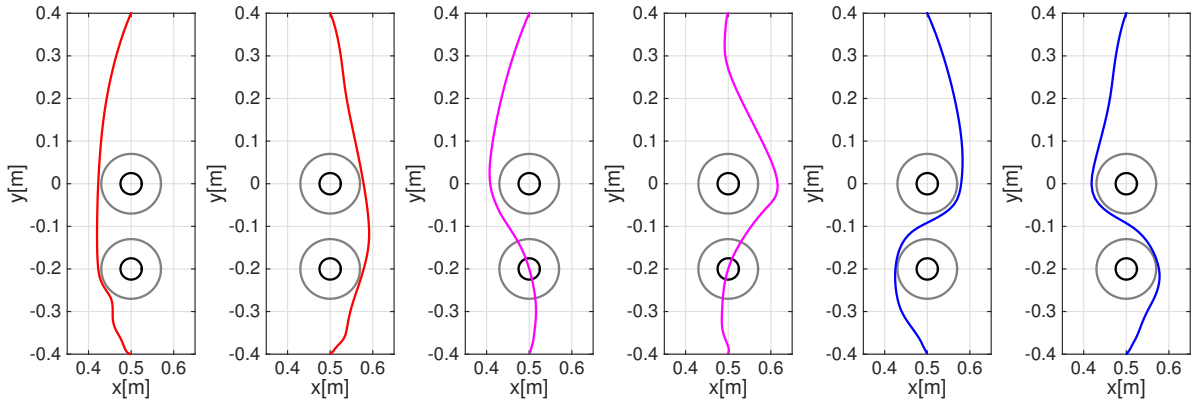


Figure 6.11: Schematic visualization of some representative trajectories obtained for the real-robot experiment using the reward in (6.18). The bottles (in black) have been expanded with a safety threshold corresponding to the width of the arm's end-effector (gray). The left-most plots show trajectories with a low reward (< -4) due to the robot not crossing in-between the bottles. The center plots with trajectories in magenta show solutions with a reward between $(-3, -2)$, while examples of quasi-optimal trajectories are shown in blue on the right-most plots, corresponding to a reward (> -1).

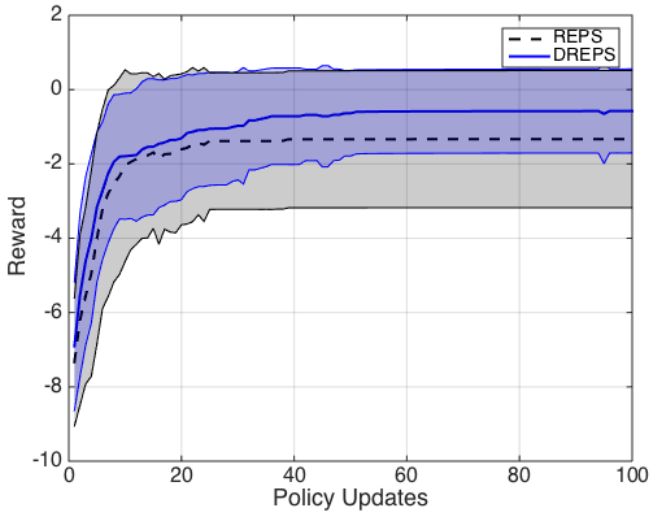


Figure 6.12: Learning curves for REPS and DREPS with the reward function in (6.18). Policy updates were calculated after every 50 rollouts.

and the learning curves with the mean and 2-standard deviations can be seen in Fig. 6.12. REPS obtained a satisfactory solution in 35 out of 50 experiments, while DREPS solved the problem correctly in 47 of them. Moreover, the average reward for the unsatisfactory solutions obtained by REPS was -4.12, while the average reward for the unsatisfactory solutions obtained by DREPS was -2.80. This is due to the fact that REPS is more likely to prematurely converge to one of the two left-most solutions in Fig. 6.11, while DREPS's repulsive term pushes the solutions away from those, which actually yield a lower reward than the magenta solutions in the middle of Fig. 6.11.

Last, we compared some sample trajectories obtained with the human-guided learning with those obtained with REPS and DREPS in Fig. 6.13. While the Human-guided trajectories seem to be less smooth, they are obtained with only 20 sample trajectories, which shows how human assistance can drastically reduce the number of real-robot samples needed in other non-simulable experiments:

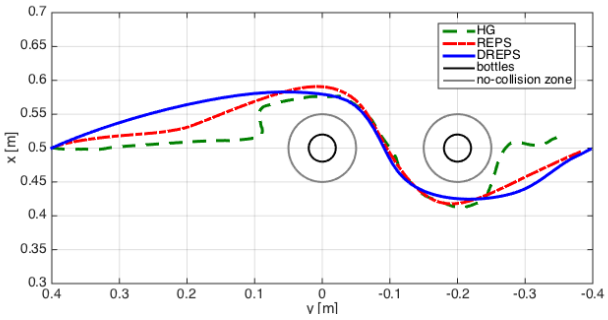


Figure 6.13: Comparison of three different ways of obtaining the solution to the bottle avoidance problem.

Regarding the computational time of the clustering, Alg. 6.1 took an average of $0.52s$ in an $i5\text{-}2400S$ CPU at 2.50GHz, clustering 50 samples of dimension 20. Such additional computational cost, together with the cost of Alg. 6.2, makes our approach more CPU-demanding than REPS. However, real robot motion is more time demanding and costly than such computational time every N robot motions. In particular, looking at Fig. 6.10, such increment on computing time results in a better learning curve, thus requiring less real-robot experiments. A video comparing REPS and DREPS in the bottle avoiding task can be found in Appendix B.5, together with the execution on the real robot of the final trajectory found by DREPS.

Unimodal task example

Moreover, we tested the performance of DREPS on a unimodal reward task, namely drawing a circle, in which the same 7-DoF WAM robot had to improve an initial motion towards a 3D circle-

tracking motion. We kinesthetically taught a 7-DoF WAM arm to follow a circular trajectory in the Cartesian space with its wrist. The circle best fitting the initial trajectory, which was very inaccurate, was computed and a cost function consisting in a point-to-point deviation from that circle, plus an acceleration-penalizing term, was considered. We fitted the taught trajectory with a Dynamic Movement Primitive (DMP). 12 Gaussians equally spaced in time were used for each DoF, as the trajectory to be learned was a complex 20-second movement. This generated a set of 84 parameters, representing the linear multipliers of such Gaussians. After applying both REPS and DREPS algorithms, the outcome after 50 learning experiments was very similar, which could be expected due to the uni-modality of the problem. The learning curves for both REPS and DREPS can be seen in Fig. 6.14. As theoretically anticipated in Section 6.1.2, DREPS displayed the same behavior as REPS in terms of learning speed and resulting trajectories.

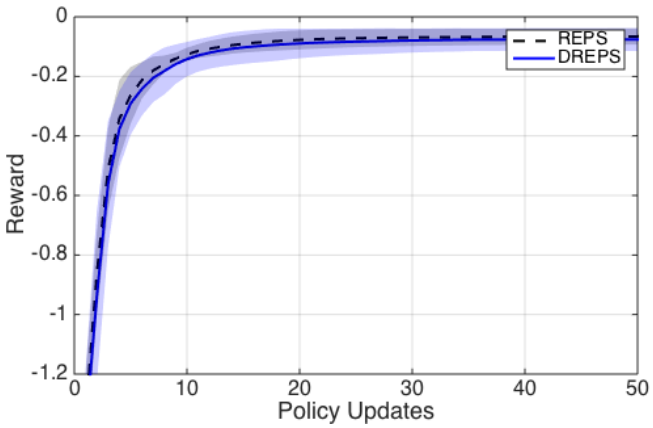


Figure 6.14: Learning curves for REPS and DREPS for a simulated unimodal problem with real robot data. Both algorithms show the same performance as expected for a unimodal task

6.2 Summary

In this chapter, we developed a generalization of the Policy Search (PS) algorithm known as Relative Entropy Policy Search. Such generalization, which is equal to REPS if the clusterization is omitted, considers the possibility of using both bad experiences to have a repulsive effect, and best data to encourage approaching the best-performing areas. This helps to influence the solution away from bad data collected during sampling/experimentation. While the performance of REPS and DREPS is similar in purely convex problems, our algorithm shows to be effective at preventing the loss of time in plateaus by other algorithms, as seen in the learning curve in Fig. 6.7, without the need of using a multi-modal solution as in Hierarchical REPS [122].

Clusterization prior to the application of DREPS as presented in this chapter has proved effective, but future work includes a deeper study of this topic, testing if other clustering algorithms or approaches to obtain the attractive and repulsive clusters can yield better results.

The proposed algorithm, while showing a very similar behaviour to REPS in uni-modal problems, is very suitable in cases where there is a multi-modal solution to the problem, but the user only needs a single solution. Multi-modal PS approaches would need more samples in order to fit the different possible solutions, while DREPS focuses on a single solution and refines it faster. We have first assessed the benefits of using DREPS using a synthetic multi-modal reward function. Then, experiments in a real robot setup have been performed, using DMPs to parametrize robot motion [134] in a task with a multi-modal reward function, and the results confirm the better performance of DREPS versus REPS.

In the following chapter, we tackle the improvement of PS efficiency from another perspective, namely motion encoding compactness. We will perform reward-oriented dimensionality reduction on the parametrizations of motion that will allow for a faster learning and also a more intuitive representation of motion through coupling of DoF.

7

Reward-oriented Dimensionality Reduction with Movement Primitives

As mentioned in Chapter 5, Movement Primitives are nowadays widely used as movement parametrization for learning robot trajectories, because of their linearity in the parameters, rescaling robustness and continuity. However, when learning a movement with MPs, a very large number of Gaussian approximations needs to be performed. Adding them up for all joints yields too many parameters to be explored when using Reinforcement Learning (RL), thus requiring a prohibitive number of experiments/simulations to converge to a solution with a (locally or globally) optimal reward. In this chapter, we address the process of simultaneously learning a MP-characterized robot motion and its underlying joint couplings through linear Dimensionality Reduction (DR), which will provide valuable qualitative information leading to a reduced and intuitive algebraic description of such motion.

These motor/motion behaviors are usually represented with Movement Primitives (MPs), introduced in Chapter 5.2. A desired trajectory is represented by fitting certain parameters, which can then be used to improve or change it. In the robotic case, such trajectories can either be in the joint domain, or in the robot's operational space/Cartesian space domain. In the latter, a proper inverse kinematics algorithm to translate Cartesian trajectories to joint motions is often needed, as discussed in Chapter 3. Those trajectories are then tracked with a proper controller, such as the one presented in Chapter 4.

In Chapter 5, we defined two types of MP: DMPs and ProMPs. Both motion characterizations result in linear policies wrt. the parameters. This linearity is not only very suitable for RL algorithms, such as Policy Search (PS), but also allows for linear dimensionality reduction techniques on the movement primitives. PS algorithms require several rollouts to find a proper policy update. In addition, to have a good fitting of the initial movement, many parameters are required, while we want to have few in order to reduce the dimensionality of the optimization

problem. When applying learning algorithms using MPs, several aspects must be taken into account:

- **Model availability**. RL can be performed through simulation or with a real robot. The first case is more practical when a good simulator of the robot and its environment is available. However, in the case of manipulation of non-rigid objects or, more generally, when accurate models are not available, reducing the number of parameters and rollouts is critical.

- **Exploration Constraints**. Certain exploration values might result in dangerous motion of the real robot, such as strong oscillations and abrupt acceleration changes. Moreover, certain tasks may not depend on all the Degrees of Freedom (DoF) of the robot, meaning that the RL algorithm used might be exploring motions that are irrelevant to the task, as we will see later.

- **Parameter dimensionality**. Despite MPs having less parameters than other motion representations, complex robots still require many parameters for a proper trajectory representation. The number of parameters needed strongly depends on the trajectory length or speed. In a 7-DoF robot following a long 20-second trajectory, the use of more than 20 Gaussian kernels per joint might be necessary, thus having at least 140 parameters in total. A higher number of parameters will usually allow for a better fitting of the initial motion characterization, but performing exploration for learning with such a high dimensional space will result in a slower learning. Therefore, there is a trade-off between better exploitation (many parameters) and efficient exploration (fewer parameters).

For these reasons, performing Dimensionality Reduction (DR) on the MPs' DoF is an effective way of dealing with the trade-off between exploitation and exploration in the parameter space to obtain a compact and descriptive projection matrix which helps the RL algorithm to converge faster to a (possibly) better solution. Additionally, Policy Search approaches in robotics usually have few sample experiments to update their policy. This results in policy updates where there are less samples than parameters, thus providing solutions with exploration covariance matrices that are rank-deficient (note that a covariance matrix obtained by linear combination of samples can't have a higher rank than the number of samples itself). These matrices are usually then regularized by adding a small value to the diagonal so the matrix remains invertible. This procedure is a greedy approach, since the unknown subspace of the parameter space is given a residual exploration value. Other approaches like Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [135] filter the covariance through the optimization updates, progressively fading the effect of old data in the covariance matrix, such data being replaced by the new data

acquired. Furthermore, a known initial covariance matrix can be used to improve the sample-based estimation, or by limiting the entropy in the covariance matrix update [136]. However, the number of samples necessary in order to properly fit a covariance matrix is usually around 4 times its dimension [135]. Therefore, performing DR in the parameter space results in a possible elimination of unexplored space, as much more information is needed in order to characterize all the dimensions in such dN_f- dimensional space. On the contrary, if such DR is performed in the DoF space, the number of samples is larger than the number of DoF of the robot and, therefore, by eliminatint one degree of freedom of the robot (or a linear combination of them), we are removing a subspace of the parameter space in which we do have information, and it will not affect the unexplored space, but rather a subspace of the DoF of the robot that is very likely to have a negligible impact on the outcome of the task.

This chapter is divided into three sections. Sections 7.1 and 7.2 present linear DR frameworks for the two types of MP presented in Chapter 5: ProMPs and DMPs, respectively. Section 7.3 presents a summary of the chapter.

7.1 Dimensionaliy Reduction for ProMPs

Humans as well as humanoid robots can use a large number of degrees of freedom to solve very complex motor tasks. The high-dimensionality of these motor tasks adds difficulties to the control problem and machine learning algorithms. In this section, we want to apply Dimensionality Reduction (DR) techniques to Probabilistic Movement Primitives (ProMPs), introduced in Section 5.2.2. While ProMPs have been shown to have many benefits, they suffer with the high-dimensionality of a robotic system as the number of parameters of a ProMP scales quadratically with the dimensionality. We use probabilistic dimensionality reduction techniques, using Expectation Maximization (EM) to extract the unknown synergies from a given set of demonstrations, while maximizing the log-likelihood function with respect to such demonstrated motions, or the reward-weighted robot exploratory executions. The ProMP representation is now estimated in the low-dimensional space of the synergies. We show that our dimensionality reduction is more efficient both for encoding a trajectory from data and for applying Reinforcement Learning with Relative Entropy Policy Search (REPS), introduced in Section 5.1.1.

Dimensionality reduction over the DoF of robots is a common approach for grasping and hand motion [137–139]. However, it has been less used for arm or leg robot skill coordination. As the curse of dimensionality affects the performance of most RL algorithms when applied to robots with a relatively high number of DoF, such as the NAO robot in Fig. 7.1, new approaches to Reinforcement Learning (RL) with Dimensionality Reduction (DR) are being developed [27, 139]. Both approaches are based on Principal Component Analysis (PCA) or its probabilistic

version (PPCA). In this chapter, we directly extract the latent space with EM, without requiring PCA. PCA is just used as initialization of the EM approach.

Figure 7.1: NAO robot interactuating with people. Picture provided by the Ave Maria Foundation in Sitges.

Using EM instead of PCA in combination with the ProMP approach comes with an important benefit, since the ProMP provides a time-dependent variance profile. While the variance profile is important in many applications, it also contains relevant information for the dimensionality reduction technique. Time points with a low variance are important for the movement and the PCA approach is not allowed to distort these time points. However, for time points with a large variance, a larger reproduction error of the DR technique is acceptable.

7.1.1 Representing Dimensionality Reduction for ProMP (DR-ProMP)

In Section 5.2.2, we introduced the ProMP representation for robot motion. However, such motion representation needs to be adapted to encapsulate the linear DR proposed. For this reason, given a robot with d DoF, we will reduce the dimensionality of its motion representation to a latent space of dimension r, which is manually given. We can express the robot's state vector $\mathbf{y}_t$ with latent space variables $\mathbf{x}_t$ using a projection matrix $\mathbf{\Omega}$ as:

$$\mathbf{y}_t \simeq \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \mathbf{\Omega}_2 \mathbf{x}_t, \tag{7.1}$$

where $\mathbf{\Omega}_s = I_s \otimes \mathbf{\Omega}$ $(sd \times sr)$, will be used throughout this chapter as the Kronecker product of the s-dimensional identity matrix and what we define as coordination matrix $\mathbf{\Omega}$ $(d \times r)$, a

linear mapping from an r-dimensional latent virtual joint space into the d-dimensional robot joint space. With this notation, we will also simplify $\mathbf{\Omega} = \mathbf{\Omega}_1$. In Eq. (7.1), we used $\mathbf{\Omega}_2$ to project both position and velocity.

In order to represent the trajectory as a linear combination of some parameters $\boldsymbol{\omega}$ as in ProMPs, we can write (7.1) as

$$\mathbf{y}_t = \mathbf{\Omega}_2 \mathbf{x}_t + \boldsymbol{\epsilon}_{fit} = \mathbf{\Omega}_2 \left(\mathbf{\Phi}_t^T \boldsymbol{\omega} + \boldsymbol{\epsilon}_x \right) + \boldsymbol{\epsilon}_{fit}, \tag{7.2}$$

with $\mathbf{\Phi}_t = [\phi_t, \dot{\phi}_t]$ being the $n \times 2$ matrix with the kernels used for the trajectory, and $\boldsymbol{\epsilon}_{fit}, \boldsymbol{\epsilon}_x$ the DR fitting error and the Gaussian noise for $\mathbf{x}$, respectively.

Thus, the probability of being in the latent state $\mathbf{x}_t$ given the weights $\boldsymbol{\omega} = [\boldsymbol{\omega}_1^T, ..., \boldsymbol{\omega}_r^T]^T$, $(rn \times 1)$ is

$$\begin{aligned} p(\mathbf{x}_t|\boldsymbol{\omega}) = & \left(\begin{bmatrix} \mathbf{x}_{1,t} \\ ... \\ \mathbf{x}_{d,t} \end{bmatrix} \middle| \begin{bmatrix} \mathbf{\Phi}_t^T & ... & \mathbf{0} \\ ... & ... & ... \\ \mathbf{0} & ... & \mathbf{\Phi}_t^T \end{bmatrix} \boldsymbol{\omega}, \mathbf{\Sigma}_x \right) \\ = & \ \mathcal{N}(\mathbf{x}_t|\mathbf{\Psi}_t^T \boldsymbol{\omega}, \mathbf{\Sigma}_x), \end{aligned} \tag{7.3}$$

with $\mathbf{\Psi}_t^T = I_r \otimes \mathbf{\Phi}_t^T$ $(2r \times rn)$. The probability of observing $\mathbf{y}_t$ given $\mathbf{x}_t$ is given by Eq. (7.2), i.e.,

$$p(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t|\mathbf{\Omega}_2 \mathbf{x}_t, \mathbf{\Sigma}_{fit}), \tag{7.4}$$

$\mathbf{\Sigma}_{fit}$ being the covariance of the reprojection error. Thus, the trajectory distribution in the full configuration space $\mathbf{y}_t$ is now

$$p(\mathbf{y}_t; \boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{y}_t|\mathbf{\Omega}_2 \mathbf{\Psi}_t^T \boldsymbol{\omega}, \mathbf{\Sigma}_y) \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega) d\boldsymbol{\omega}, \tag{7.5}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega, \mathbf{\Omega}, \mathbf{\Sigma}_y\}$ is the set of parameters of the DR-ProMP representation, and $\mathbf{\Sigma}_y = \mathbf{\Sigma}_{fit} + \mathbf{\Omega}_2 \mathbf{\Sigma}_x \mathbf{\Omega}_2^T$ the total system noise. We next define how Eq. (7.5) generalizes to the case of context variables.

Given a context variable $\mathbf{s} \in \mathbb{R}^M$, where M is the number of context variables, we can extend the ProMP framework to contextual ProMP, $p(\boldsymbol{\omega}|\mathbf{s}) = \mathcal{N}\left(\boldsymbol{\omega}|\boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s}, \mathbf{\Sigma}_\omega\right)$, thus (7.5) becomes:

$$p(\mathbf{y}_t|\mathbf{s}; \boldsymbol{\theta}) = \int \mathcal{N}(\mathbf{y}_t|\mathbf{\Omega}_2 \mathbf{\Psi}_t^T \boldsymbol{\omega}, \mathbf{\Sigma}_y) \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s}, \mathbf{\Sigma}_\omega) d\boldsymbol{\omega},$$

where $\mathbf{K}_\omega$ is a linear mapping between the context variables and the ProMP mean.

7.1.2 DR-ProMP for robot control

In order to fully exploit the DR-ProMP, we must also characterize a stochastic controller that reproduces the motion variance. Assuming a known discrete-time linearized dynamics of the system with a time step of dt, the robot's dynamics equation is

$$\mathbf{y}_{t+dt} = (\mathbf{I} + dt\mathbf{A}_t)\mathbf{y}_t + \mathbf{B}_t dt\mathbf{u} + \mathbf{c}_t dt, \tag{7.6}$$

where $\mathbf{A_t}$, $\mathbf{B_t}$, and $\mathbf{c_t}$ are the system, input and drift terms of the first-order Taylor expansion of the dynamical system of the robot. This translates to a reduced dimensionality dynamic equation as

$$\mathbf{\Omega}_2\mathbf{x}_{t+dt} = (\mathbf{I} + dt\mathbf{A}_t)\mathbf{\Omega}_2\mathbf{x}_t + \mathbf{B}_t dt\mathbf{u} + \mathbf{c}_t dt. \tag{7.7}$$

Hence,

$$\mathbf{x}_{t+dt} = \mathbf{\Omega}_2^\dagger(\mathbf{I} + dt\mathbf{A}_t)\mathbf{\Omega}_2\mathbf{x}_t + \mathbf{\Omega}_2^\dagger\mathbf{B}_t dt\mathbf{u} + \mathbf{\Omega}_2^\dagger\mathbf{c}_t dt, \tag{7.8}$$

where $\dagger$ is used for the Moore-Penrose pseudoinverse. However, in Eq. (7.8), we still have a control action defined in the full $d-$dimensional space. Then, we can act on the reduced dimension $r-$space by using the control signal $\mathbf{u} \in \mathbb{R}^d$

$$\mathbf{u} = \mathbf{\Omega}\boldsymbol{\nu} + \boldsymbol{\epsilon}_u,$$

where $\boldsymbol{\epsilon}_u \sim \mathcal{N}(0, \mathbf{\Sigma}_u/dt)$ is defined as in [127]. The latent space controller $\boldsymbol{\nu} \in \mathbb{R}^r$ is defined with a linear gain $\mathbf{K}_t$ and drift $\boldsymbol{\kappa}_t$

$$\boldsymbol{\nu} = \mathbf{K}_t\mathbf{x}_t + \boldsymbol{\kappa}_t. \tag{7.9}$$

We will keep the controller noise in the high-dimensional space, allowing for exploration also outside the latent space.

Thus, inserting (7.9) into (7.8) results in

$$\begin{aligned} \mathbf{x}_{t+dt} = \; & \mathbf{\Omega}_2^\dagger\left[(\mathbf{I} + dt\mathbf{A}_t)\mathbf{\Omega}_2 + dt\mathbf{B}_t\mathbf{\Omega}\mathbf{K}_t\right]\mathbf{x}_t + \\ & \mathbf{\Omega}_2^\dagger\mathbf{B}_t(\mathbf{\Omega}\boldsymbol{\nu} + \boldsymbol{\epsilon}_u)dt + \mathbf{\Omega}_2^\dagger\mathbf{c}_t dt \\ = \; & \mathbf{F}\mathbf{x}_t + \mathbf{f} + \mathbf{\Omega}_2^\dagger\mathbf{B}_t dt\boldsymbol{\epsilon}_u, \end{aligned} \tag{7.10}$$

where $\mathbf{F} = \mathbf{\Omega}_2^\dagger\left[(\mathbf{I} + dt\mathbf{A}_t)\mathbf{\Omega}_2 + dt\mathbf{B}_t\mathbf{\Omega}\mathbf{K}_t\right]$ and $\mathbf{f} = dt\mathbf{\Omega}_2^\dagger\left(\mathbf{B}_t\mathbf{\Omega}\boldsymbol{\nu}dt + \mathbf{\Omega}_2^\dagger\mathbf{c}_t\right)$. Given the dynamics

Equation (7.10), we can extract the probability of being in state $\mathbf{x}_{t+dt}$ at the next time step, i.e.,

$$p(\mathbf{x}_{t+dt}) = \int \mathcal{N}(\mathbf{x}_{t+dt}|\mathbf{F}\mathbf{x}_t + \mathbf{f}, \mathbf{\Sigma}_s dt)\mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)d\mathbf{x}_t = \\ \mathcal{N}(\mathbf{x}_{t+dt}|\mathbf{F}\boldsymbol{\mu}_t + \mathbf{f}, \mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_s dt), \tag{7.11}$$

with $\boldsymbol{\mu}_t = \mathbf{\Psi}_t^T \boldsymbol{\mu}_\omega$ and $\mathbf{\Sigma}_t = \mathbf{\Psi}_t^T \mathbf{\Sigma}_\omega \mathbf{\Psi}_t$. We can match the control noise matrix $\mathbf{\Sigma}_s$ for each timestep by using the cross-correlation between consecutive steps of the trajectory distribution [127] and obtain

$$dt\mathbf{\Sigma}_s = \mathbf{\Sigma}_{t+dt} - \mathbf{C}_t^T \mathbf{\Sigma}_t \mathbf{C}_t,$$

with $\mathbf{C}_t = \mathbf{\Psi}_t^T \mathbf{\Sigma}_\omega \mathbf{\Psi}_{t+dt}$. The controller terms $\mathbf{K}_t$, $\boldsymbol{\kappa}_t$ can be obtained by matching $\{\boldsymbol{\mu}_{t+dt}, \mathbf{\Sigma}_{t+dt}\}$ from the system dynamics in Eq. (7.11) and the ProMP model. See [127] for more details of a similar deduction,

$$\mathbf{K}_t = \mathbf{\Omega}^T \mathbf{B}^\dagger \left[\mathbf{\Omega}_2 \left(\dot{\boldsymbol{\psi}}_t^T \mathbf{\Sigma}_\omega \dot{\boldsymbol{\psi}}_t - \mathbf{\Sigma}_s/2\right) - \mathbf{A}_t \mathbf{\Omega}_2 \mathbf{\Sigma}_t\right] \mathbf{\Sigma}_t^{-1},$$

and

$$\boldsymbol{\kappa}_t = \mathbf{\Omega}^\dagger \mathbf{B}^\dagger \left[\mathbf{\Omega}_2 \left(\dot{\boldsymbol{\psi}}_t^T \boldsymbol{\mu}_\omega - \mathbf{A}_t \mathbf{\Omega}_2 + \mathbf{B}_t \mathbf{\Omega} \mathbf{K}_t\right) \boldsymbol{\psi}_t^T \boldsymbol{\mu}_\omega - \mathbf{c}_t\right],$$

and the controller noise covariance estimation is given by

$$\mathbf{\Sigma}_u = \mathbf{B}^\dagger \mathbf{\Omega}_2 \mathbf{\Sigma}_s \mathbf{\Omega}_2^T \mathbf{B}^{T\dagger} + \alpha^2 \mathbf{I}. \tag{7.12}$$

Note that, defined as in (7.12), $\mathbf{\Sigma}_u$ would be a $d \times d$ symmetric, semipositive definite matrix with rank r at most, corresponding to that of the latent subspace defined by $\mathbf{\Omega}$. Adding the term $\alpha\mathbf{I}$, for a small value of α will ensure we can explore the neighborhood of the latent space when executing the ProMP.

7.1.3 Fitting DR-ProMP parameters with expectation maximization

We can estimate the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega, \mathbf{\Omega}, \mathbf{\Sigma}_y, \mathbf{K}_\omega\}$ in closed form using EM. $\mathbf{\Psi}_t^T$ will now be a $(r \times rN_f)$ matrix, and $\mathbf{y}_t^k$ will be a d-dimensional vector. We will use the vector $\mathbf{Y}^k$ to denote the concatenated position vectors of a single trajectory k,

$$\mathbf{Y}^k = \left[\mathbf{y}_1^{kT}, ..., \mathbf{y}_{N_t}^{k\ T}\right]^T.$$

For our EM-algorithm, we will consider the most general case of a contextual ProMP in the latent space $\mathbf{x}$. Hence, we need to estimate the following parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega, \mathbf{\Omega}, \mathbf{\Sigma}_y, \mathbf{K}_\omega\}$.

Additionally, we want to use our model estimation algorithm for policy search algorithms that are based on data re-weighting. These algorithms introduce a weighting d_k for each trajectory. Hence, we also have to consider such a weighting in our EM algorithm. In this section, the most general case will be considered for obtaining the ProMP parameters.

For a context-free ProMP, $\mathbf{K}_\omega$ can be set to zero, while the weights d_k can be set to 1 when all demonstrations have the same importance.

We will maximize the weighted marginal log-likelihood $\sum_k d_k \log p(\mathbf{y}_t|s,\boldsymbol{\theta})$ of the data and the latent space representation, thus we have to derive the equations with the marginalized $\boldsymbol{\omega}$ with the difficulties it entails. The following subsections explain how to obtain the log-likelihood function and differentiate it.

Expectation step

In the expectation step, we must find the probabilities for each demonstration k with the old parameters $\boldsymbol{\theta}^{old}$:

$$p(\boldsymbol{\omega}|\mathbf{Y}^k) = \frac{p(\mathbf{Y}^k|\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}^k)} \propto p(\mathbf{Y}^k|\boldsymbol{\omega})p(\boldsymbol{\omega}), \tag{7.13}$$

where, using $\boldsymbol{\Omega}_{N_t} = \mathbf{I}_{N_t} \otimes \boldsymbol{\Omega}$,

$$p(\mathbf{Y}^k|\boldsymbol{\omega}) = \mathcal{N}(\mathbf{Y}^k|\boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\omega}, \mathbf{I}_{N_t} \otimes \boldsymbol{\Sigma}_y). \tag{7.14}$$

Note that in the contextual case, we have omitted the conditioning on the context variables for simplicity of the equations. Using the Bayes rule for Gaussian distributions, see Equations (39) and (40) in [140], we obtain:

$$p(\boldsymbol{\omega}|\mathbf{Y}^k) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s}_k + \boldsymbol{\Sigma}_\omega \boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T \boldsymbol{\Gamma}^{-1}(\mathbf{Y}^k - \boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T)(\boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s}_k)$$

$$\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_\omega - \boldsymbol{\Sigma}_\omega \boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T \boldsymbol{\Gamma}^{-1}\boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\Sigma}_\omega,$$

and $\boldsymbol{\Gamma}$ is given by $\boldsymbol{\Gamma} = \mathbf{I}_{N_t} \otimes \boldsymbol{\Sigma}_y + \boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T$.

Maximization step

Given the posterior probabilities $p(\boldsymbol{\omega}|\mathbf{Y}^k)$ for each demonstration, we now maximize the weighted expectation of the log-likelihood function, where d_k is used as weight for each trajectory, i.e.,

$$L = \sum_{k=1}^{N_d} d_k \mathbb{E}_{\boldsymbol{\omega}|\mathbf{Y}^K;\boldsymbol{\theta}^{old}} \left[\log\Big(p(\boldsymbol{\omega}, \mathbf{Y}^k; \boldsymbol{\theta})\Big)\right]$$

$$= \sum_{k=1}^{N_d} d_k \int_{\boldsymbol{\omega}} p(\boldsymbol{\omega}|\mathbf{Y}^k; \boldsymbol{\theta}^{old}) \log \Big(p(\mathbf{Y}^k|\boldsymbol{\omega}) p(\boldsymbol{\omega}) \Big) \, d\boldsymbol{\omega},$$

where

$$\log p(\mathbf{Y}^k|\boldsymbol{\omega}) p(\boldsymbol{\omega}) = \log p(\boldsymbol{\omega}) + \sum_{t=1}^{N_t} \log p(\mathbf{y}_t^k|\boldsymbol{\omega}),$$

with $p(\boldsymbol{\omega})$ defined as in (5.17) and $p(\mathbf{y}_t^k|\boldsymbol{\omega}) = \mathcal{N}(\mathbf{y}_t^k|\boldsymbol{\Omega}\boldsymbol{\Psi}^T\boldsymbol{\omega}, \boldsymbol{\Sigma}_y)$.

Then, using the expectation identities for linear and quadratic transformations, (31) and (33) in [140], we obtain the value of the likelihood function to maximize in the M-step:

$$L = -\tfrac{1}{2}\left[\left(\sum_{k=1}^{N_d} d_k\right) \log|2\pi\boldsymbol{\Sigma}_\omega| + N_t \log|2\pi\boldsymbol{\Sigma}_y|\right] - \tfrac{1}{2}\sum_{k=1}^{N_d} d_k (\boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s} - \boldsymbol{\mu}_k)\boldsymbol{\Sigma}_\omega^{-1}(\boldsymbol{\mu}_\omega + \mathbf{K}_\omega \mathbf{s} - \boldsymbol{\mu}_k)$$
$$-\tfrac{1}{2}\sum_{k=1}^{N_d} d_k \sum_{t=1}^{N_t} \left[(y_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_y^{-1}(y_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k) + \mathrm{tr}(\boldsymbol{\Psi}_t\boldsymbol{\Omega}^T\boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_k)\right] - \frac{1}{2}\sum_{k=1}^{N_d} d_k \mathrm{tr}(\boldsymbol{\Sigma}_\omega^{-1}\boldsymbol{\Sigma}_k).$$

Now, we can derivate L w.r.t. each of the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega^{-1}, \boldsymbol{\Omega}, \boldsymbol{\Sigma}_y^{-1}, \mathbf{K}_\omega\}$. Using that $\Delta_A \mathrm{tr}(ABA^TC) = CAB + C^TAB^T$, (see Section 8 in [141]), that $\lambda = tr(\lambda)$ for scalar values, the invariance of the trace w.r.t. cyclic permutations, the derivative of the log of a determinant, the derivative of a product in a trace [141] and the derivative w.r.t. the transpose of a matrix, we can obtain a closed-form solution of the parameters by setting each derivative to zero

$$\boldsymbol{\mu}_\omega = \left(\sum_{k=1}^{N_d} d_k\right)^{-1} \sum_{k=1}^{N_d} d_k(\boldsymbol{\mu}_k - \mathbf{K}_\omega \mathbf{s}), \tag{7.15}$$

$$\boldsymbol{\Sigma}_\omega = \left(\sum_{k=1}^{N_d} d_k\right)^{-1} \sum_{k=1}^{N_d} d_k \left[\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)(\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)^T\right], \tag{7.16}$$

$$\boldsymbol{\Omega} = \left[\sum_{k=1}^{N_d} d_k \sum_{t=1}^{N_t} y_t^k (\boldsymbol{\mu}_k^T \boldsymbol{\Psi}_t)\right] \left[\sum_{t=1}^{N_t} \boldsymbol{\Psi}_t^T \sum_{k=1}^{N_d} d_k \left(\boldsymbol{\Sigma}_k + \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T\right) \boldsymbol{\Psi}_t\right]^\dagger, \tag{7.17}$$

$$\mathbf{K}_\omega = \left[\sum_{k=1}^{N_d} d_k(\boldsymbol{\mu}_k - \boldsymbol{\mu}_\omega)\mathbf{s}_k^T\right] \left[\sum_{k=1}^{N_d} d_k \mathbf{s}_k \mathbf{s}_k^T\right]^\dagger, \tag{7.18}$$

$$\begin{aligned}\boldsymbol{\Sigma}_y = \left(\sum_{k=1}^{N_d} d_k\right)^{-1} \frac{1}{N_t} \sum_{k=1}^{N_d}\sum_{t=1}^{N_t} d_k[\boldsymbol{\Omega}\boldsymbol{\Psi}_t^T \boldsymbol{\Sigma}_k \boldsymbol{\Psi}_t \boldsymbol{\Omega}^T \\ +(\mathbf{y}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T \boldsymbol{\mu}_k)(\mathbf{y}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T \boldsymbol{\mu}_k)^T].\end{aligned} \tag{7.19}$$

In order to update all the parameters in the M-step, we will first obtain the new mean for the weights from Eq. (7.15), and subsequently use it to obtain its covariance with Eq. (7.16). As the next step, we compute the new coordination matrix $\boldsymbol{\Omega}^{new}$ with (7.17) and $\mathbf{K}_\omega$ in the case of contextual ProMP with (7.18). Finally, we use all the newly computed parameters to obtain the new noise covariance $\boldsymbol{\Sigma}_y$ with (7.19).

Initialization

Given a fixed value $r \leq d$ of the latent space dimension, a good initialization of the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Omega}, \boldsymbol{\Sigma}_y, \mathbf{K}_\omega\}$ can increase the convergence speed of the EM algorithm. To that purpose, we will perform PCA on the trajectories $\mathbf{Y}^k$ of the robot for all demonstrations $k = 1..N_d$. After obtaining an initial guess for $\boldsymbol{\Omega}$ and use it in Eq. (7.2), we obtain the fitting weights $\boldsymbol{\omega}_k$ for each demonstration k, which we will use to initialize $\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega$

$$\boldsymbol{\mu}_\omega = \frac{1}{N_d}\sum_{k=1}^{N_d} \boldsymbol{\omega}_k,$$

$$\boldsymbol{\Sigma}_\omega = \frac{1}{N_d}\sum_{k=1}^{N_d} (\boldsymbol{\omega}_k - \boldsymbol{\mu}_\omega - \mathbf{K}_\omega \mathbf{s})(\boldsymbol{\omega}_k - \boldsymbol{\mu}_\omega - \mathbf{K}_\omega \mathbf{s})^T.$$

Finally, we can initialize $\boldsymbol{\Sigma}_y$ as

$$\boldsymbol{\Sigma}_y = \frac{1}{N_d N_t}\sum_{k=1}^{N_d}\sum_{t=1}^{N_t} (\mathbf{y}_t^k - \overline{\mathbf{y}})(\mathbf{y}_t^k - \overline{\mathbf{y}})^T,$$

where $\overline{y}$ is the average over all timesteps and demonstrations of the joint state vector.

In the case of a contextual ProMP, we will initialize $[\boldsymbol{\mu}_\omega, \mathbf{K}_\omega]$ together using weighted least squares with the weight vector $\mathbf{d} = [d_1, ..., d_{N_d}]$

$$[\boldsymbol{\mu}_\omega, \mathbf{K}_\omega]^T = \left(\mathbf{S}\text{diag}(\mathbf{d})\mathbf{S}^T + \lambda\mathbf{I}\right)^{-1} \mathbf{S}^T \text{diag}(\mathbf{d})\mathbf{W}_d^T, \tag{7.20}$$

where $\mathbf{W}_d = [\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_{N_d}]$ are the weights obtained from fitting the demonstrations, $\lambda\mathbf{I}$ a regularization term, $\mathbf{d}$ the weights vector for each demonstration and $\mathbf{S}$ is a matrix containing

all the context vectors for the demonstrations

$$\mathbf{S} = \begin{bmatrix} 1 & \dots & 1 \\ \mathbf{s}_1 & \dots & \mathbf{s}_{N_d} \end{bmatrix},$$

where the 1s in the first row are added to be able to simultaneously compute $\boldsymbol{\mu}_\omega$ and $\mathbf{K}_\omega$.

Comparison of EM versus PCA

To illustrate the benefits of the EM-based algorithm presented in this section in comparison to PCA, we created d-dimensional probabilistic trajectories, tracked with a stochastic controller. We fitted these trajectories using both a PCA matrix projection and our EM-based approach and used the *Kullbach-Leibler divergence* [120]. We computed the KL divergence between the data distribution and the fitted models using EM and PCA. In Fig. 7.2, we can see the color plot of the ratio ρ, defining the relative KL-divergence gain w.r.t. the PCA approach for a set of simulated ProMP and their DR fitting. We can observe that, despite not being significantly better when the fitting dimension is equal or almost equal to the original dimension, the KL-divergence of the original trajectory distribution fitted with our approach is reduced by around 20% over the PCA approach with ρ defined as

$$\rho = \frac{\sum_{t=1}^{N_t} \text{KL}(p(y_t; \text{data}) | p(y_t; \boldsymbol{\theta}^{em})) - \text{KL}(p(y_t; \text{data}) | p(y_t; \boldsymbol{\theta}^{pca}))}{\sum_{t=1}^{N_t} \text{KL}(p(y_t; \text{data}) | p(y_t; \boldsymbol{\theta}^{pca}))}, \tag{7.21}$$

Reinforcement Learnig with DR-ProMP

The proposed EM algorithm can be straightforwardly used for RL algorithms that are based on data re-weighting [12]. These algorithms are used to choose the weighting d_k of each data trajectory. We will use the REPS algorithm described in Section 5.1.1 The weights d_k provided by REPS for the set of N_k trajectories are used to infer a new policy by using weighted maximum likelihood estimation. In our case, we will use our EM algorithm for learning synergetic ProMP to obtain a new policy. In the case of contextual variables, a contextual version of REPS can also be found in [12].

After observing a set of a trajectories, and given their relative importance (weights) provided by the REPS algorithm, we can use the EM estimation in Section 7.1.3 to update the parameters of the ProMP using Equations (7.15)-(7.19). In Alg.7.1, we show the iterative procedure for learning with the EM-REPS approach. This algorithm updates all the ProMP parameters given the reward-based weights of the executed trajectories. If no context variables are considered, $\mathbf{K}_\omega$

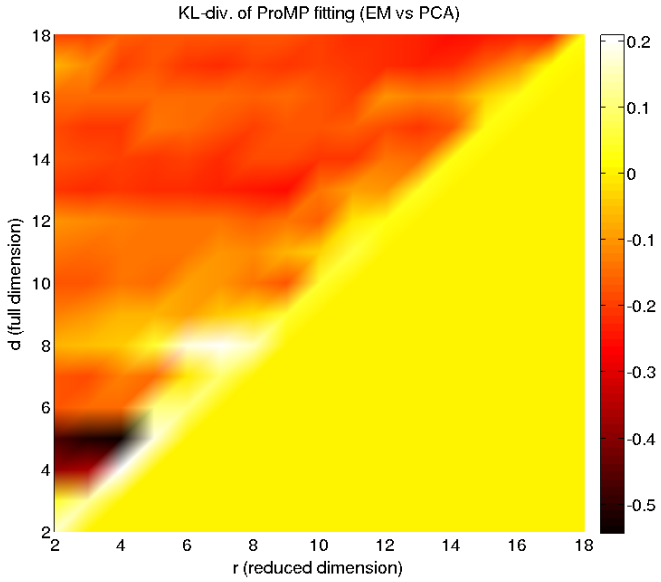


Figure 7.2: Plot of the ratio ρ defined in (7.21), showing that the EM approach reduces the KL-divergence by around 20% w.r.t. PCA.

Algorithm 7.1: EM-REPS learning with DR-ProMP

Input:
Previous DR-ProMP parameters $\boldsymbol{\theta}^{old} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Omega}, \boldsymbol{\Sigma}_y, \mathbf{K}_\omega, \}$.
Kullback-Liebler divergence bound ϵ_{kl}.
Other ProMP parameters dt, N_t, N_f.

1: **for** $k = 1...N_k$ **do**
2: Obtain the context variable $\mathbf{s}_k$.
3: Reproduce ProMP sampling from the trajectory distribution, store joint data $\mathbf{Y}^k$ and evaluate reward function R_k.
4: **end for**
5: Compute weights using contextual REPS: $d_k = \text{reps.}(\mathbf{R}, \epsilon_{kl})$
6: **while** no convergence **do**
7: Perform weighted EM in Section 7.1.3 to obtain new parameters $\boldsymbol{\theta}^{\text{new}} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega \boldsymbol{\Omega}, \boldsymbol{\Sigma}_y, \mathbf{K}_\omega\}$.
8: **end while**

and $\mathbf{s}$ can be ignored from Equations (7.15)-(7.19), and use the non-contextual REPS in [121].

7.1.4 Experiments

We performed two experiments to evaluate the proposed approach, a first one fitting a lower-dimensionality walking policy for the NAO robot, and a comparison of methods for RL with DMP and REPS on a planar manipulator.

Walking couplings of a NAO robot

We used our DR-ProMP approach to encapsulate similar walking behaviors of the robot *NAO* in Fig. 7.1. The trajectories for its leg joints, which are 6-DoF for each leg, were obtained by controlling the Zero Moment Point of the robot [142]. In Fig. 7.4, we show the observed distribution (in red) obtained from 13 different walking experiments and its fit with the proposed method (in blue) for a reduced dimensionality of $r = 4 \leq 12 = d$. We can clearly see that 4 synergies are enough to represent the whole walking behavior. We can extract its couplings and relations from the matrix $\mathbf{\Omega}$, which relates the joints according to the correlation matrix plotted in Fig. 7.3. Note that the intuitive couplings between joints are those arising from a symmetry of the legs position and thus, joints $< 1, 7 >$, $< 3, 9 >$, $< 4, 10 >$, $< 5, 11 >$ will usually have opposite values, while $< 2, 8 >$, $< 6, 12 >$ would have the same sign. In fact, joints $< 1, 7 >$ are mechanically coupled, so the effective dimensionality is $d = 11$.

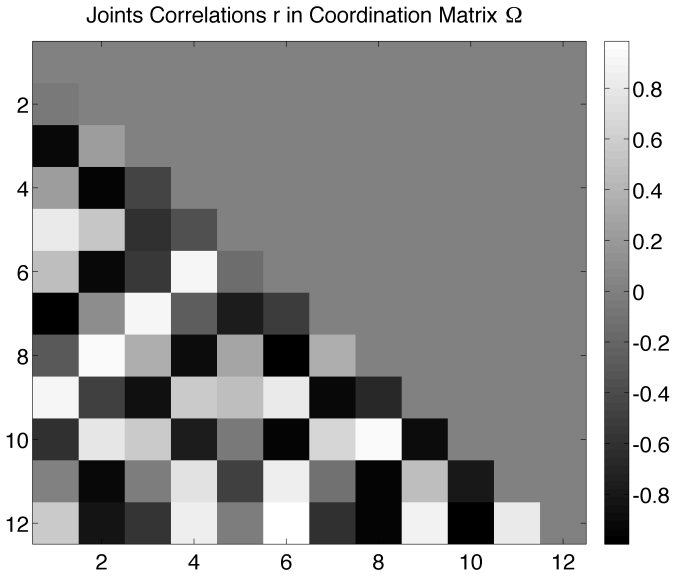


Figure 7.3: Correlation between joints provided by the coordination matrix $\mathbf{\Omega}$. Black color indicates $r \sim -1$ while white indicates $r \sim 1$.

Planar trajectory tracking

We simulated a $d = 15$ DoF planar arm, with the all joints having a length of 1 m. The task was to follow a Cartesian trajectory.

Initialization and reward function Using the same initial and desired conditions for all experiments, we compared the following learning strategies:

- *DMP+REPS*. We obtained the weights for each demonstration with least squares, and fitted

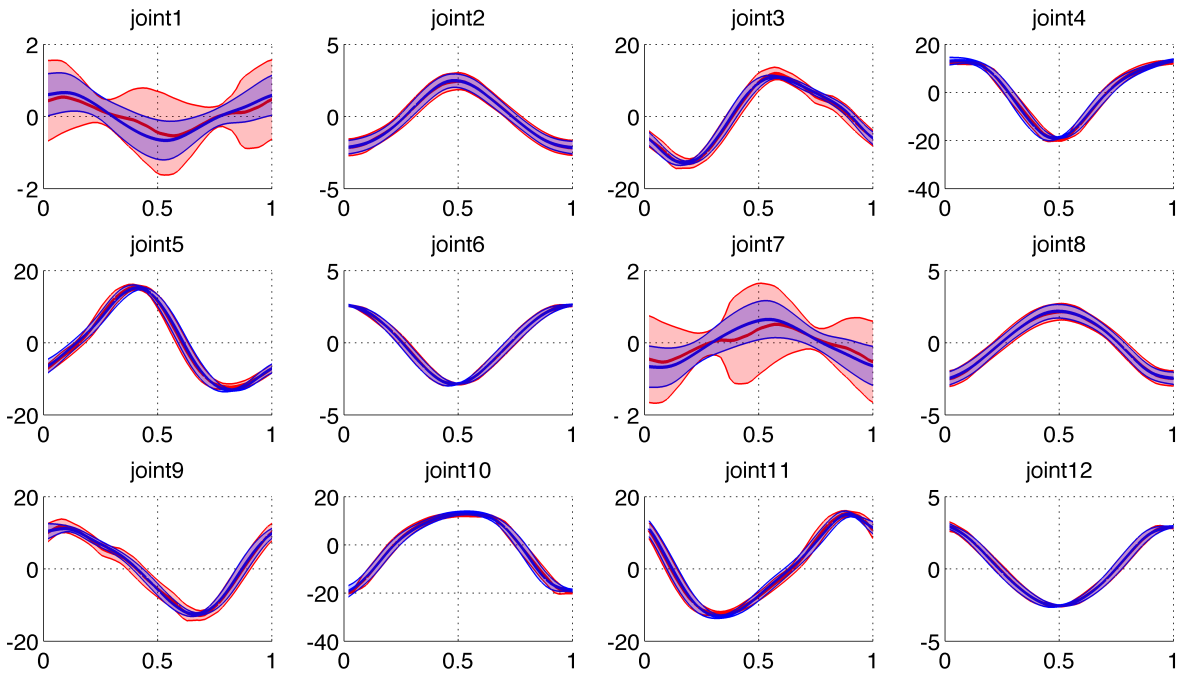


Figure 7.4: Original walking trajectory distribution, in degrees, for each one of the 12 NAO leg joints (red) and the fitting obtained with a reduced dimension of 4 (blue) in a normalized time-scale. We can see that, despite the dimensionality reduction, the trajectory distribution could be reproduced accurately.

a distribution $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$ over them, which was used for sampling and exploring with REPS.

- *ProMP+REPS*.

- *EM-ProMP+REPS* as in Alg. 7.1 with the full dimension.

- *EM-ProMP+REPS* as in Alg. 7.1 with $r = 8$ and no perturbation ($\alpha = 0$) on $\boldsymbol{\Omega}$.

- *EM-ProMP+REPS* as in Alg. 7.1 with $r = 8$ and a perturbation of $\alpha = 0.01$ on $\boldsymbol{\Omega}$.

To keep the possibility of exploring outside the restricted joint subspace provided by the projection matrix $\boldsymbol{\Omega}$, we will add a perturbation $\epsilon_\alpha \sim \mathcal{N}(0, \alpha^2)$ to each element of the coordination matrix before every rollout, which will provide a similar effect to the α value given in Eq. (7.12). We used 100 trajectory samples, and updated the policy every 20 samples once we had the first 100. For the REPS algorithm, we took a Kullback-Leibler bound of $\epsilon_{KL} = 0.5$. We used 8 Gaussian kernel functions for each DoF, up to a total of $8d$ or $8r$ for the reduced dimension cases. We performed 100 policy updates and the reward function was given by

$$R = \sum_{t=1}^{N_t} - \left(\mathbf{e}_t^T \mathbf{D} \mathbf{e}_t + \ddot{\mathbf{q}}_t^T \mathbf{H} \ddot{\mathbf{q}}_t \right), \tag{7.22}$$

where $\mathbf{D}$ and $\mathbf{H}$ are diagonal matrices. To generate samples with ProMP, the desired framework would be to use the stochastic controller defined in Section 7.1.2. However, the controller is computationally more expensive and can be sensitive to numerical conditioning of the covariances involved for a small timestep. For these reasons, we sampled the trajectories by directly evaluating $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$.

Results and discussion

In Fig. 7.5, we display the average and standard deviation over 10 experiments for each algorithm. The results show that, for $r = d$, the EM-based algorithm does not improve over the standard REPS update. This behavior was expected as, for the full-rank case of the coordination matrix, the parameter update mostly becomes equivalent to the standard REPS case, with the addition of possible numerical error. However, the DR on the ProMP (black line in the plot) slightly improves performance over the previous algorithm. This improvement becoming more substantial when a perturbation α is added to the coordination matrix, yielding also a faster improvement on the earlier updates, due to the reduced parameter dimensionality. For the DMP case, we observe a more unstable behavior in the earlier steps but better average (with more variance) on reward than with the standard ProMP approach.

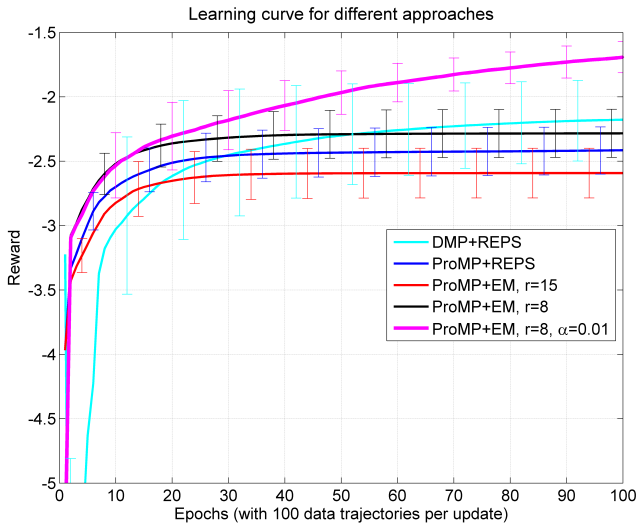


Figure 7.5: Comparing different learning approaches on a 15-DoF planar manipulator simulated task.

7.1.5 Conclusions

In Section 7.1.1, we provided a novel, EM-based approach to fit trajectory distributions with a ProMP that computes a linear latent space for the DoF of the robot. Working in this latent space with a small perturbation on the coordination matrix or the controller provides a faster algorithm for RL with REPS, thanks to the drastic reduction of the parameters associated to those eliminated DoF. Our future aim is to automatically estimate the optimal space dimension r and improve the computational cost of evaluating the posterior of the expectation step in Section 7.1.3. In the following section, we will apply a similar approach to another kind of MP: Dynamic Movement Primitives (DMPs), introduced in Section 5.2.1.

7.2 Dimensionality Reduction for DMPs

In the previous section, we developed a linear Dimensionality Reduction (DR) technique for ProMPs. Such approach can also be applied to other MPs, such as Dynamic Movement Primitives (DMPs). DMPs are probably the most popular MPs used in robotics nowadays, as already presented in Section 5.2.1. However, DMPs also suffer from the curse of dimensionality when learning tasks with a high dimension of the parameter space. For this reason, the DR technique is also very suitable in this case. In this section, we address the process of simultaneously learning a DMP-characterized robot motion and its underlying joint couplings, also through linear Dimensionality Reduction (DR), which will provide valuable qualitative information leading to

a reduced and intuitive algebraic description of such motion. The results in the experimental section show that not only can we effectively perform DR on DMPs while learning, but we can also obtain better learning curves. We will present the alternatives to reduce the parameter dimensionality of the DMP characterization in Section 7.2.1, focusing on the robot's DoF and using principal component analysis to perform DR. In Section 7.2.2, we will use EM to compute such couplings as we applied to ProMPs. Then, experimental results with a simulated planar robot, a single 7-DoF WAM robot, and a bimanual task performed by two WAM robots will be discussed in Section 7.2.3, followed by conclusions and future work prospects in Section 7.2.4.

7.2.1 DMP coordination

In this section, we will describe how to efficiently obtain the joint couplings associated to each task during the learning process, in order to both reduce the dimensionality of a problem, as well as obtaining a linear mapping describing a task. In [143, 144], a coordination framework for DMPs was presented, where a robot's movement primitives were coupled through a coordination matrix, which was learned with a RL algorithm. Kormushev et al. [145] worked in a similar direction, using square matrices to couple d primitives represented as attractor points in the task space domain.

We now propose to use a not necessarily square coordination matrix in order to decrease the number of actuated DoF and thus reduce the number of parameters. Recalling Eq. (5.14):

$$\begin{aligned} \ddot{\mathbf{y}}/\tau^2 &= \alpha_z \left(\beta_z \left(\mathbf{y}_g - \mathbf{y}\right) - \dot{\mathbf{y}}/\tau\right) + \mathbf{f}(x) \\ \mathbf{f}(x) &= \boldsymbol{\Psi}_t^T \boldsymbol{\omega}, \end{aligned} \tag{7.23}$$

We can add a term $\boldsymbol{\Omega}$ in the excitation term $\mathbf{f}(x)$:

$$\mathbf{f}(x_t) = \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T \boldsymbol{\omega}, \tag{7.24}$$

for each timestep t, $\boldsymbol{\Omega}$ being a $(d \times r)$ matrix, with $r \leq d$ a reduced dimensionality, $\boldsymbol{\Psi}_t^T = \mathbf{I}_r \otimes \mathbf{g}$, similarly as in the previous section, and $\boldsymbol{\omega}$ is an (rN_f)-dimensional vector of motion parameters. Note that this representation is equivalent to having r movement primitives encoding the d-dimensional acceleration command vector $\mathbf{f}(x)$. Intuitively, the columns of $\boldsymbol{\Omega}$ represent the couplings between the robot's DoF.

The DR reduction in Eq. (7.24) is preferable to a DR on the DMP parameters themselves for numerical reasons. If such DR would be performed as $\mathbf{f}(x_t) = \boldsymbol{\Psi}_t^T \hat{\boldsymbol{\Omega}} \boldsymbol{\omega}$, then $\hat{\boldsymbol{\Omega}}$ would be a high-dimensional matrix but, more importantly, the number of rollouts per policy update performed in PS algorithms would determine the maximum dimension of the explored space as a subspace

of the parameter space, leaving the rest of such parameter space with zero value or a small regularization value at most. In other words, performing DR in the parameter space requires N_f times more rollouts per update to provide full information than performing such DR in the joint space.

In order to learn the coordination matrix $\mathbf{\Omega}$, we need an initial guess and also an algorithm to update it and eliminate unnecessary degrees of freedom from the DMP, according to the reward/cost obtained. Within this representation, we can assume that the probability of having certain excitation values $\mathbf{f}_t = \mathbf{f}(x_t)$ at a timestep given the weights $\boldsymbol{\omega}$ is $p(\mathbf{f}_t|\boldsymbol{\omega}) \sim \mathcal{N}(\mathbf{\Omega}\mathbf{\Psi}_t^T\boldsymbol{\omega}, \mathbf{\Sigma}_\text{f})$, $\mathbf{\Sigma}_\text{f}$ being the system noise. Thus, if $\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\omega}, \mathbf{\Sigma}_\omega)$, the probability of $\mathbf{f}_t$ is:

$$p(\mathbf{f}_t) = \mathcal{N}(\mathbf{\Omega}\mathbf{\Psi}_t^T\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\text{f} + \mathbf{\Omega}\mathbf{\Psi}_t^T\mathbf{\Sigma}_\omega\mathbf{\Psi}_t\mathbf{\Omega}^T). \tag{7.25}$$

Along this Section we will firstly present the initialization of such coordination matrices, and how they can be updated with a reward-aware procedure. Additionally, we present ways of eliminating robot DoF irrelevant for a certain task, and a multiple coordination matrix framework to segment a trajectory so as to use more than one projection matrices. Finally, we consider some numerical issues and a summary of the variants defined.

Obtaining an initial coordination matrix with PCA

In this section, we will explain how to obtain the coordination motion matrices while learning a robotic task, and how to update them. A proper initialization for the coordination matrix $\mathbf{\Omega}$ is to perform a Principal Component Analysis (PCA) over the demonstrated values of $\mathbf{f}$ (see Eq. (5.14)). Taking the matrix $\overline{\mathbf{F}}$ of all timesteps $\mathbf{f}_t$ in Eq. (7.24), of size $(d \times N_t)$, for the d degrees of freedom and N_t timesteps as:

$$\overline{\mathbf{F}} = \begin{bmatrix} f_\text{de}^{(1)}(x_0) - \overline{f}_\text{de}^{(1)} & \dots & f_\text{de}^{(1)}(x_{N_t}) - \overline{f}_\text{de}^{(1)} \\ \dots & & \dots \\ f_\text{de}^{(d)}(x_0) - \overline{f}_\text{de}^{(d)} & \dots & f_\text{de}^{(d)}(x_{N_t}) - \overline{f}_\text{de}^{(d)} \end{bmatrix}, \tag{7.26}$$

$\overline{f}_\text{de}$ being the average over each joint component of the DMP excitation function, for the demonstrated motion (de subindex). Then we can perform Singular Value Decomposition (SVD), obtaining $\overline{\mathbf{F}} = \mathbf{U}_{pca}\mathbf{\Sigma}_{pca}\mathbf{V}_{pca}^T$.

Now having set $r < d$ as a fixed value, we can take the r eigenvectors with the highest singular values, which will be the first r columns of $U_{pca} = [\mathbf{u}_1, ..., \mathbf{u}_r, ..., \mathbf{u}_d]$, with associated

singular values $\sigma_1 > \sigma_2 > ... > \sigma_d$ and use

$$\mathbf{\Omega} = [\mathbf{u}_1, ..., \mathbf{u}_r] \tag{7.27}$$

as coordination matrix in Eq. (7.24), having a reduced set of DoF of dimension r, which activate the robot joints (dimension d), minimizing the error in the reprojection $e = \|\overline{\mathbf{F}} - \mathbf{\Omega}\overline{\mathbf{\Sigma}}\mathbf{V}_{pca}^T\|_{Frob}^2$, with $\overline{\mathbf{\Sigma}}$ the part of $\mathbf{\Sigma}_{pca}$ corresponding to the first r singular values.

Note that this dimensionality reduction does not take any reward/cost function into consideration, so an alternative would be to start with a full-rank coordination matrix and progressively reduce its dimension, according to the costs or rewards of the rollouts. In the next section, we will explain the methodology to update such coordination matrix while also reducing its dimensionality, if necessary.

Reward-based Coordination Matrix Update (CMU)

In order to tune the coordination matrix once initialized, we assume we have performed N_k reproductions of motion, namely rollouts, obtaining an excitation function $\mathbf{f}_t^{(j),k}$, for each rollout $k = 1..N_k$, timestep $t = 1..N_t$, and DoF $j = 1..d$. Now having evaluated each of the trajectories performed with a cost/reward function, we can also associate a relative weight P_t^k to each rollout and timestep as it is done in policy search algorithms such as PI2 or REPS. We can then obtain a new $N_t \times d$ matrix $\mathbf{F}_{\text{co}}$ with the excitation function on all timesteps defined as:

$$\mathbf{F}_{\text{co}}^{new} = \begin{bmatrix} \sum_{k=1}^{N_k} \mathbf{f}_1^{(1),k} P_1^k & ... & \sum_{k=1}^{N_k} \mathbf{f}_{N_t}^{(1),k} P_{N_t}^k \\ ... & & ... \\ \sum_{k=1}^{N_k} \mathbf{f}_1^{(d),k} P_1^k & ... & \sum_{k=1}^{N_k} \mathbf{f}_{N_t}^{(d),k} P_{N_t}^k \end{bmatrix}, \tag{7.28}$$

which is a $(d \times N_t)$ matrix containing information of the excitation functions, weighted by their relative importance according to the rollout result. A new coordination matrix $\mathbf{\Omega}$ can be obtained by means of PCA. However, when changing the coordination matrix, we then need to refit the parameters $\{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega\}$ to make the trajectory representation fit the same trajectory. To this end, given the *old* distribution (represented with a hat) and the one with the new coordination matrix, the excitation functions distributions, excluding the system noise, are

$$\hat{\mathbf{f}}_t \sim \mathcal{N}(\hat{\mathbf{\Omega}}\hat{\mathbf{\Psi}}_t^T\hat{\boldsymbol{\mu}}_\omega, \hat{\mathbf{\Omega}}\hat{\mathbf{\Psi}}_t^T\hat{\mathbf{\Sigma}}_\omega\hat{\mathbf{\Psi}}_t\hat{\mathbf{\Omega}}^T) \tag{7.29}$$

$$\mathbf{f}_t \sim \mathcal{N}(\mathbf{\Omega}\mathbf{\Psi}_t^T\boldsymbol{\mu}_\omega, \mathbf{\Omega}\mathbf{\Psi}_t^T\mathbf{\Sigma}_\omega\mathbf{\Psi}_t\mathbf{\Omega}^T). \tag{7.30}$$

We also represent the whole trajectories:

$$\mathcal{F} = \begin{bmatrix} \mathbf{f}_1 \\ ... \\ \mathbf{f}_{N_t} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{O}\boldsymbol{\Psi}^T\boldsymbol{\mu}_\omega, \mathbf{O}\boldsymbol{\Psi}^T\boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}\mathbf{O}^T\right), \tag{7.31}$$

where $\mathbf{O} = \mathbf{I}_{N_t} \otimes \boldsymbol{\Omega}$, and

$$\boldsymbol{\Psi} = \begin{bmatrix} \mathbf{I}_r \otimes \mathbf{g}_1^T \\ ... \\ \mathbf{I}_r \otimes \mathbf{g}_{N_t}^T \end{bmatrix}, \tag{7.32}$$

while

$$\hat{\mathcal{F}} = \begin{bmatrix} \hat{\mathbf{f}}_1 \\ ... \\ \hat{\mathbf{f}}_{N_t} \end{bmatrix} \sim \mathcal{N}\left(\hat{\mathbf{O}}\hat{\boldsymbol{\Psi}}^T\hat{\boldsymbol{\mu}}_\omega, \hat{\mathbf{O}}\hat{\boldsymbol{\Psi}}^T\hat{\boldsymbol{\Sigma}}_\omega\hat{\boldsymbol{\Psi}}\hat{\mathbf{O}}^T\right), \tag{7.33}$$

where $\hat{\mathbf{O}} = \mathbf{I}_{N_t} \otimes \hat{\boldsymbol{\Omega}}$, and $\hat{\boldsymbol{\Psi}}$ is built accordingly to the value of r in case the dimension has changed, as it will be seen later.

To minimize the loss of information when updating the distribution parameters $\boldsymbol{\mu}_\omega$ and $\boldsymbol{\Sigma}_\omega$, given a new coordination matrix, we can minimize the Kullbach-Leibler (KL) divergence between $\hat{p} \sim \mathcal{N}(\hat{\boldsymbol{\mu}_\omega}, \hat{\boldsymbol{\Sigma}_\omega})$ and $p \sim \mathcal{N}(\mathbf{M}\boldsymbol{\mu}_\omega, \mathbf{M}\boldsymbol{\Sigma}_\omega\mathbf{M}^T)$, being $\mathbf{M} = (\hat{\mathbf{O}}\hat{\boldsymbol{\Psi}}^T)^\dagger\mathbf{O}\boldsymbol{\Psi}^T$, $\dagger$ representing the Moore-Penrose pseudoinverse operator. This reformulation is done so we have two probability distributions with the same dimensions and considering the non-symmetry of the KL divergence, knowing that $\mathbf{f}_t$ is used to approximate $\hat{\mathbf{f}}_t$.

As the KL divergence for two normal distributions is known [140], we have

$$\text{KL}(\hat{p}\|p) = \log\frac{|\mathbf{M}\boldsymbol{\Sigma}_\omega\mathbf{M}^T|}{|\hat{\Sigma}_\omega|} + \text{tr}\left((\mathbf{M}\boldsymbol{\Sigma}_\omega\mathbf{M}^T)^{-1}\hat{\boldsymbol{\Sigma}}_\omega\right) + (\mathbf{M}\boldsymbol{\mu}_\omega - \hat{\boldsymbol{\mu}}_\omega)^T(\mathbf{M}\boldsymbol{\Sigma}_\omega\mathbf{M}^T)^{-1}(\mathbf{M}\boldsymbol{\mu}_\omega - \hat{\boldsymbol{\mu}}_\omega) - d \tag{7.34}$$

Now, derivating wrt. $\boldsymbol{\mu}_\omega$ and wrt. $(\mathbf{M}\boldsymbol{\Sigma}_\omega\mathbf{M}^T)^{-1}$, and setting the derivative to zero to obtain the minimum, we obtain:

$$\boldsymbol{\mu}_\omega = \mathbf{M}^\dagger\hat{\boldsymbol{\mu}}_\omega \tag{7.35}$$

$$\boldsymbol{\Sigma}_\omega = \mathbf{M}^\dagger\left[\hat{\boldsymbol{\Sigma}}_\omega + (\mathbf{M}\boldsymbol{\mu}_\omega - \hat{\boldsymbol{\mu}}_\omega)(\mathbf{M}\boldsymbol{\mu}_\omega - \hat{\boldsymbol{\mu}}_\omega)^T\right](\mathbf{M}^T)^\dagger. \tag{7.36}$$

Minimizing the KL divergence provides the solution with the least loss of information, in terms of probability distribution on the excitation function.

Algorithm 7.2: Coordination Matrix Update (CMU)

Input:
Rollout and timestep probabilities P_t^k, $k = 1..N_k$, $t = 1..N_t$.
Excitation function $\mathbf{f}_i^{(j),k}$, $j = 1..d$.
Previous update (or initial) excitation function $\mathbf{F}_{\text{co}}$.
Current $\mathbf{\Omega}$ of dimension $d \times r$.
DoF discarding threshold η.
Current DMP parameters $\boldsymbol{\theta} = \{\mathbf{\Omega}, \boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega\}$.

1: Compute $\mathbf{F}_{\text{co}}^{\text{new}}$ as in Eq. (7.28)
2: Filter excitation matrix: $\mathbf{F}_{\text{co}}^{\text{new}} = \alpha \mathbf{F}_{\text{co}}^{\text{new}} + (1-\alpha)\mathbf{F}_{\text{co}}$
3: Subtract average as in Eq. (7.26)
4: Perform PCA and obtain $U_{pca} = [\mathbf{u}_1, ..., \mathbf{u}_r, ..., \mathbf{u}_d]$ (see Eqs. (7.26)(7.27))
5: **if** $\sigma_1/\sigma_r > \eta$ **then**
6: $\quad r = r - 1$
7: **end if**
8: $\mathbf{\Omega}^{new} = [\mathbf{u}_1, ..., \mathbf{u}_r]$
9: Recompute: $\{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega\}$ as in Eqs. (7.35)-(7.36)

Eliminating irrelevant degrees of freedom

In RL, the task the robot tries to learn does not always necessarily depend on all the degrees of freedom of the robot. For example, if we want to track a Cartesian xyz position with a 7-DoF robot, it is likely that some degrees of freedom, which mainly alter the end-effector's orientation, may not affect the outcome of the task. However, these DoF are still considered all through the learning process, causing unnecessary motions which may slow down the learning process or generate a final solution in which a part of the motion was not necessary.

For this reason, the authors claim that the main use of a coordination matrix should be to remove those unnecessary degrees of freedom, and the coordination matrix, as built throughout this section, can easily provide such result. Given a threshold η for the ratio of the maximum and minimum singular values of $\mathbf{F}_{\text{co}}^{\text{new}}$ defined in Eq. (7.28), we can discard the last column of the coordination matrix if those singular values verify $\sigma_1/\sigma_r > \eta$.

In Algorithm 7.2, we show the process of updating and reducing the coordination matrix, where the parameter α is a filtering term, in order to keep information from previous updates.

Multiple Coordination Matrix Update (MCMU)

Using a coordination matrix to translate the robot degrees of freedom into others more relevant to task performance may result in a too strong linearization. For this reason, multiple coordination matrices can be built in order to perform a coordination framework that uses different

mappings throughout the trajectory. In order to do so, we will use a second layer of N_s Gaussians and build a coordination matrix $\mathbf{\Omega}_s$ for each Gaussian $s = 1..N_s$, so that at each timestep the coordination matrix $\mathbf{\Omega}^t$ will be an interpolation between such constant coordination matrices $\mathbf{\Omega}_s$. To compute such an approximation, linear interpolation of projection matrices does not necessarily yield robust result. For that reason, given the time t and the constant matrices $\mathbf{\Omega}_s$, we compute

$$\mathbf{\Omega}^t = \text{argmax}_{\mathbf{X}} \sum_{s=1}^{N_s} \varphi_s^t \left[\text{tr}(\mathbf{\Omega}_s^\dagger \mathbf{X}) - d \log \left(\frac{\|\mathbf{X}\|_F}{\|\mathbf{\Omega}_s\|_F} \right) \right] \tag{7.37}$$

being

$$\varphi_s^t = \varphi_s(x_t) = \frac{\phi_s(x_t)}{\sum_{p=1}^{N_s} \phi_p(x_t)}, \tag{7.38}$$

where ϕ_s, $s = 1..N_s$ are equally distributed Gaussians in the time domain, and $\|.\|_F$ is the Frobenius norm. A new Gaussian basis function set is used in order to independently choose the number of coordination matrices, as the amount of Gaussian kernels for the DMPs are usually much larger than the number needed for linearizing the trajectory in the robot's DoF domain. Such number N_s then can be arbitrarily set, according to the needs and variability of the trajectory. The optimization cost is chosen for its similarity with the covariance terms of the Kullback-Leibler divergence, and if we use the number of DoF of the robot, d, as a factor in the equation and the matrices $\mathbf{\Omega}_s$ are all orthonormal, then the optimal solution is a linear combination of such matrices:

$$\mathbf{\Omega}^t = \sum_{s=1}^{N_s} \varphi_s^t \mathbf{\Omega}_s. \tag{7.39}$$

Note that φ_s^t acts as an activator for the different matrices, but it is also used to distribute the responsibility for the acceleration commands to different coordination matrices in the newly-computed matrix $\mathbf{F}_{co}^s$. Then we can proceed as in the previous section, with the exception that we will compute each $\mathbf{\Omega}_s$ independently by using the following data for fitting:

$$\mathbf{F}_{co}^s = \begin{bmatrix} \sum_{k=1}^{N_k} \mathbf{f}_1^{(1),k} \varphi_s^1 P_1^k & \dots & \sum_{k=1}^{N_k} \mathbf{f}_{N_t}^{(1),k} w_1^s P_{N_t}^k \\ \dots & & \dots \\ \sum_{k=1}^{N_k} \mathbf{f}_1^{(d),k} \varphi_s^{N_t} P_1^k & \dots & \sum_{k=1}^{N_k} \mathbf{f}_{N_t}^{(d),k} \varphi_s^{N_t} P_{N_t}^k \end{bmatrix}, \tag{7.40}$$

and use the following excitation function in Eq. (5.14):

$$\mathbf{f}(x) = \mathbf{\Omega}_t \mathbf{\Psi}_t^T \boldsymbol{\mu}_\omega, = \left(\sum_{s=1}^{N_s} \varphi_s^t \mathbf{\Omega}_s \right) \mathbf{\Psi}_t^T \boldsymbol{\mu}_\omega. \tag{7.41}$$

Now, changing the linear relation between the d robot degrees of freedom and the r variables encoding them within a probability distribution (see Eq. (7.25)) requires that we update the covariance matrix in order to keep it consistent with the different coordination matrices. In this case, as $\mathbf{\Omega}$ is varying, we can reproject the weights similarly as in Eqs. (7.29)-(7.36), by using:

$$\boldsymbol{O} = \begin{bmatrix} \mathbf{\Omega}^1 & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & \mathbf{\Omega}^{N_t} \end{bmatrix}, \tag{7.42}$$

for the new values of r, $\mathbf{\Omega}_s$, $\forall s$, compared to the previous values (now denoted with a hat). We can then use Eqs. (7.35) and (7.36) to recalculate $\boldsymbol{\mu}_\omega$ and $\mathbf{\Sigma}_\omega$.

Numerical issues of a sum of two coordination matrices

While using several projection matrices $\mathbf{\Omega}_s$, $s = 1..N_s$ has the advantage of providing more flexibility for representing variable motions and allows to reduce the dimensionality further, some numerical aspects need to be taken into account:

Orthogonality of components and locality

When using Eq. (7.39) to define the coordination matrix, we are in fact doing a weighted sum of them, obtaining a matrix whose *j*-th column is the weighted sum of the *j*-ths columns of the N_s different coordination matrices. This operation would not necessarily provide a matrix with its columns pairwise orthonormal, despite all the $\mathbf{\Omega}_s$ having that property. However, this property is not needed other than to have an easier-to compute inverse operator. The smaller the differences between consecutive coupling matrices, the closer to an orthonormal column-wise matrix we will obtain at each timestep. From this fact, we conclude that the number of coupling matrices has to be fitted to the implicit variability of the task, so as to keep consecutive coordination matrices relatively similar.

Eigenvector matching and sign

Another issue that may arise is that, when computing the singular value decomposition, some algorithms provide ambiguous representations in terms of the signs of the columns of the matrix U_{pca} in Section 7.2.1. This means that it can be the case of two coordination matrices, $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$, having similar columns with opposite signs, the resulting vector being a weighted difference between them, which will then translate into a computed coupling matrix $\mathbf{\Omega}^t$ obtained through

Algorithm 7.3: Reordering of PCA results

Input:
$\mathbf{\Omega}_s$,$\forall s = 1..N_s$, computed with PCA

1: **for** $is = 2..N_s$ **do**
2: Initialize $\mathbf{K} = \mathbf{0}_{r\times r}$, the pair-wise dot product matrix
3: Initialize PCAROT $= \mathbf{0}_{r\times r}$, the rotation matrix
4: **for** $i1 = 1..r$, $i2 = 1..r$ **do**
5: $\mathbf{K}(i1, i2) = \text{dot}(\mathbf{\Omega}_1(:, i1), \mathbf{\Omega}_s(:, i2))$
6: **end for**
7: **for** $j = 1..r$ **do**
8: $v_{max} = \max(|\mathbf{K}(:, j)|)$
9: $i_{max} = \text{argmax}(|\mathbf{K}(:, j)|)$
10: **if** $v_{max} = \max(|\mathbf{K}(i_{max}, :)|)$ **then**
11: PCAROT$(i_{max}, j) = \text{sign}(\mathbf{K}(i_{max}, j))$
12: **end if**
13: **end for**
14: **if** rank(PCAROT) $< r$ **then**
15: Return to line 7
16: **end if**
17: **end for**

Eq. (7.39) with a column vector that only represents *noise*, instead of a joint coupling.

It can also happen that consecutive coordination matrices $\mathbf{\Omega}_1$, $\mathbf{\Omega}_2$ have similar column vectors but, due to similar eigenvalues coming from the singular value decomposition, their column order becomes different.

Because of these two facts, a reordering of the coupling matrices $\mathbf{\Omega}_s$ has to be carried out, as shown in Algorithm 7.3. In such algorithm, we use the first coordination matrix $\mathbf{\Omega}_1$ as a reference and, for each other $s = 2..N_s$, we compute the pairwise column dot product of the reference $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_s$. We then reorder the eigenvectors in $\mathbf{\Omega}_s$ and change their signs according to the dot products matrices.

Variants of the DR-DMP method

To sum up the proposed dimensionality reduction methods for DMPs (DR-DMP), based on PCA and described in this section, we list their names and initialization in Tables 7.1 and 7.2, which show their descriptions and usages.

In Table 7.2, PCA(r) represents Principal Component Analysis (PCA) keeping the r eigenvectors with the largest singular values (see. Eq. (7.27)). N_s-PCA(r) is used to represent the computation of N_s PCA approximations and coordination using equally-initialized weights in Eq. (7.40). The CMU algorithm is defined in Algorithm 7.2, and its MCMU variant. In the

Table 7.1: Methods description

DR-DMP$^0(r)$	Fixed $\mathbf{\Omega}$ of dimension $(d \times r)$
DR-DMP$^0(N_s, r)$	Fixed multiple $\mathbf{\Omega}_s$ of dimension $(d \times r)$
DR-DMP$_{\text{CMU}}(r)$	Recalculated $\mathbf{\Omega}$ of dimension $(d \times r)$
DR-DMP$_{\text{MCMU}}(N_s, r)$	Recalculated multiple $\mathbf{\Omega}_s$ of dimension $(d \times r)$
IDR-DMP$_{\text{CMU}}$	Iterative DR while recalculating $\mathbf{\Omega}$
IDR-DMP$_{\text{MCMU}}(N_s)$	Iterative DR while recalculating multiple $\mathbf{\Omega}_s$

Table 7.2: Methods initialization and usage

Method	Initialization of $\mathbf{\Omega}$	$\boldsymbol{\theta}$ update
DR-DMP$^0(r)$	PCA(r)	REPS
DR-DMP$^0(r)$	N_s-PCA(r)	REPS
DR-DMP$_{CMU}(r)$	PCA(r)	REPS+CMU, $\eta = \infty$
DR-DMP$_{\text{MCMU}}(N_s, r)$	N_s-PCA(r)	REPS+MCMU, $\eta = \infty$
IDR-DMP$_{CMU}(r)$	PCA(d)	REPS+CMU, $\eta < \infty$
IDR-DMP$_{\text{MCMU}}(N_s, r)$	N_s-PCA(d)	REPS+MCMU, $\eta < \infty$

following section, we obtain the projection matrix $\mathbf{\Omega}$ by means of expectation maximization, as we did in Section 7.1 for ProMPs.

7.2.2 EM approach to find the latent space projection

Given the motion characterization in Section 7.2.1, we can also obtain the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \mathbf{\Sigma}_\omega, \mathbf{\Omega}, \mathbf{\Sigma}_f\}$ in closed form using Expectation-Maximization (EM). Using an EM approach will have the advantage of treating the DMPs as a probability distribution, contrary to the deterministic approach provided by the PCA. In this section, we will detail how to obtain such parameters, using the vector $\mathbf{F}^k$ to denote the concatenated acceleration commands of a single trajectory k,

$$\mathbf{F}^k = \left[{\mathbf{f}_1^k}^T, ..., {\mathbf{f}_{N_t}^k}^T \right]^T.$$

As we want to use our model estimation algorithm for policy search algorithms that are based on data re-weighting, we will have a weight d_k for each trajectory, obtained by a PS algorithm. Hence, we also have to consider such a weighting in our EM algorithm, which is derived similarly as with ProMPs in Section 7.1.3. The model can be initialized with the previously detailed PCA approach or with trivial values, such as the identity for the parameter covariance matrix $\mathbf{\Sigma}_\omega$ and zeros for the mean $\boldsymbol{\mu}_\omega$.

The procedure is exactly equivalent to that in Section 7.1.3; we maximize the weighted marginal log-likelihood $\sum_k d_k \log p(\mathbf{y}_t, \boldsymbol{\theta})$ of the data and the latent space representation, and

also derive the equations with the marginalized $\boldsymbol{\omega}$.

Expectation step

The probabilities for each demonstration k with the old parameters $\hat{\boldsymbol{\theta}}$ are found in the expectation step:

$$p(\boldsymbol{\omega}|\mathbf{F}^k) = \frac{p(\mathbf{F}^k|\boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{F}^k)} \propto p(\mathbf{F}^k|\boldsymbol{\omega})p(\boldsymbol{\omega}), \tag{7.43}$$

where, using $\boldsymbol{\Omega}_{N_t} = \mathbf{I}_{N_t} \otimes \boldsymbol{\Omega}$,

$$p(\mathbf{F}^k|\boldsymbol{\omega}) = \mathcal{N}(\mathbf{F}^k|\boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\omega}, \mathbf{I}_{N_t} \otimes \boldsymbol{\Sigma}_f). \tag{7.44}$$

Using the Bayes rule for Gaussian distributions, results in:

$$p(\boldsymbol{\omega}|\mathbf{F}^k) = \mathcal{N}(\boldsymbol{\omega}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_\omega + \boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T\boldsymbol{\Gamma}^{-1}(\mathbf{F}^k - \boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T(\boldsymbol{\mu}_\omega))$$

$$\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_\omega - \boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T\boldsymbol{\Gamma}^{-1}\boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\Sigma}_\omega,$$

and $\boldsymbol{\Gamma}$ is given by $\boldsymbol{\Gamma} = \mathbf{I}_{N_t} \otimes \boldsymbol{\Sigma}_f + \boldsymbol{\Omega}_{N_t}\boldsymbol{\Psi}^T\boldsymbol{\Sigma}_\omega\boldsymbol{\Psi}\boldsymbol{\Omega}_{N_t}^T$.

Maximization step

Given the probabilities $p(\boldsymbol{\omega}|\mathbf{F}^k)$ for each demonstration, we now maximize the weighted expectation of the log-likelihood function, where d_k is used as weight for each trajectory:

$$L = \sum_{k=1}^{N_d} d_k \mathbb{E}_{\boldsymbol{\omega}|\mathbf{F}^K;\hat{\boldsymbol{\theta}}}\left[\log\left(p(\boldsymbol{\omega}, \mathbf{F}^k; \boldsymbol{\theta})\right)\right]$$

$$= \sum_{k=1}^{N_d} d_k \int_{\boldsymbol{\omega}} p(\boldsymbol{\omega}|\mathbf{F}^k; \hat{\boldsymbol{\theta}}) \log\left(p(\mathbf{F}^k|\boldsymbol{\omega})p(\boldsymbol{\omega})\right) d\boldsymbol{\omega},$$

where

$$\log p(\mathbf{F}^k|\boldsymbol{\omega})p(\boldsymbol{\omega}) = \log p(\boldsymbol{\omega}) + \sum_{t=1}^{N_t} \log p(\mathbf{f}_t^k|\boldsymbol{\omega}),$$

with $p(\boldsymbol{\omega}) \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$ and $p(\mathbf{f}_t^k|\boldsymbol{\omega}) = \mathcal{N}(\mathbf{f}_t^k|\boldsymbol{\Omega}\boldsymbol{\Psi}^T\boldsymbol{\omega}, \boldsymbol{\Sigma}_f)$.

To obtain the likelihood function to maximize in the M-step:

$$L = -\tfrac{1}{2}\left[\left(\sum_{k=1}^{N_d} d_k\right)\log|2\pi\boldsymbol{\Sigma}_\omega| + N_t \log|2\pi\boldsymbol{\Sigma}_f|\right] - \tfrac{1}{2}\sum_{k=1}^{N_d} d_k(\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_\omega^{-1}(\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)$$
$$-\tfrac{1}{2}\sum_{k=1}^{N_d} d_k \sum_{t=1}^{N_t}\left[(\mathbf{f}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_f^{-1}(\mathbf{f}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k) + \mathrm{tr}(\boldsymbol{\Psi}_t\boldsymbol{\Omega}^T\boldsymbol{\Sigma}_f^{-1}\boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_k)\right] - \frac{1}{2}\sum_{k=1}^{N_d} d_k \mathrm{tr}(\boldsymbol{\Sigma}_\omega^{-1}\boldsymbol{\Sigma}_k). \tag{7.45}$$

Now, derivating L w.r.t. each of the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega, \boldsymbol{\Omega}, \boldsymbol{\Sigma}_f^{-1}, \}$, and solving Eqs. (7.45) as in Section 7.1.3, we obtain:

$$\boldsymbol{\mu}_\omega = \left(\sum_{k=1}^{N_d} d_k\right)^{-1}\sum_{k=1}^{N_d} d_k\boldsymbol{\mu}_k, \tag{7.46}$$

$$\boldsymbol{\Sigma}_\omega = \left(\sum_{k=1}^{N_d} d_k\right)^{-1}\sum_{k=1}^{N_d} d_k\left[\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)(\boldsymbol{\mu}_\omega - \boldsymbol{\mu}_k)^T\right], \tag{7.47}$$

$$\begin{aligned}\boldsymbol{\Omega} = &\left[\sum_{k=1}^{N_d} d_k \sum_{t=1}^{N_t} f_t^k(\boldsymbol{\mu}_k^T\boldsymbol{\Psi}_t)\right] \\ &\cdot\left[\sum_{t=1}^{N_t}\boldsymbol{\Psi}_t^T\sum_{k=1}^{N_d} d_k\left(\boldsymbol{\Sigma}_k + \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T\right)\boldsymbol{\Psi}_t\right]^\dagger,\end{aligned} \tag{7.48}$$

$$\begin{aligned}\boldsymbol{\Sigma}_f = &\left(\sum_{k=1}^{N_d} d_k\right)^{-1}\tfrac{1}{N_t}\sum_{k=1}^{N_d}\sum_{t=1}^{N_t} d_k[\boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_k\boldsymbol{\Psi}_t\boldsymbol{\Omega}^T \\ &+(\mathbf{f}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k)(\mathbf{f}_t^k - \boldsymbol{\Omega}\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_k)^T].\end{aligned} \tag{7.49}$$

With these equations, we will first obtain the new mean for the weights from Eq. (7.46), and subsequently use it to obtain its covariance with Eq. (7.47). Then, we compute the new coordination matrix $\boldsymbol{\Omega}^{new}$ with (7.48). Finally, we use all the newly computed parameters to obtain the new noise covariance $\boldsymbol{\Sigma}_f$ with (7.49).

7.2.3 Experimentation

To assess the performance of the different algorithms presented throughout Section 7.2, we performed three experiments. An initial one consisting of a fully-simulated 10-DoF planar robot, a simulated experiment with 7-DoF real robot data initialization and a real-robot experiment

with two coordinated 7-DoF robots.

10-DoF planar arm experiment

As an initial learning problem for testing, we take the planar arm task used as a benchmark in [124], where a d-dimensional planar arm robot learns to adapt an initial trajectory to go through some via-points.

Initialization and reward function

Taking $d = 10$, we generated a minimum jerk trajectory from an initial position to a goal position. As a cost function, we used the Cartesian positioning error on two via-points. The initial motion was a min-jerk-trajectory for each of the 10 joints of the planar arm robot, with each link of length $1m$, from an initial position $q_j(t = 0) = 0 \; \forall j$, to the position $q_j(t = 1) = 2\pi/d$ (see Fig. 7.6a). Then, to initialize the trajectory variability, we generated several trajectories for each joint by adding

$$q_j(t) = q_{j,minjerk}(t) + \sum_{a=1}^{2} A_a \text{exp}\left(-(t - c_a)^2/d_a^2\right),$$

where $A_a \sim \mathcal{N}(0, \frac{1}{4d})$, and obtained trajectories from a distribution as those shown for one joint in Fig. 7.6b. We used those trajectories to initialize $\boldsymbol{\omega}_\mu$ and $\boldsymbol{\Sigma}_\omega$

The task to learn is to modify the trajectory so as to go through $N_v = 2$ via points along the trajectory. As a reward function for the experiments, we used $R = \sum_t r_t$, where r_t is the time-step reward and is defined as:

$$\begin{aligned} r_t = -\sum_{v=1}^{N_v} \delta(t = t_v)(\mathbf{x}_t - \mathbf{x}_v)^T \mathbf{C}_x (\mathbf{x}_t - \mathbf{x}_v) \\ -\ddot{\mathbf{x}}_t^T \mathbf{C}_u \ddot{\mathbf{x}}_t, \end{aligned} \tag{7.50}$$

which is a weighted sum of an acceleration command and a via-points error; $\mathbf{x}_t, \mathbf{x}_v$ being the Cartesian trajectory point and via-point coordinates for each of the $1..N_v$ via-points. This cost function penalizes accelerations in the first joints, which move the whole robot. As algorithmic parameters, we used a bound on the KL-divergence of 0.5 for REPS, and a threshold $\eta = 50$ for the ratio of the maximum and minimum singular values for dimensionality reduction in Algorithm 1.

We used REPS for the learning experiment for a fixed dimension (initially set to a value $r = 1..10$), and starting with $r = 10$ and letting the algorithm reduce the dimensionality by itself. We also allowed for an exploration outside the linear subspace represented by the coordination matrix (noise added to the *d*-dimensional acceleration commands in simulation)

following $\epsilon_{noise} \sim \mathcal{N}(0, 0.1)$.

Results and discussion

After running the simulations, we obtained the results detailed in Table 7.3, where the mean and its 95% confidence interval variability are shown (through 20 runs for each case). An example of solution found can be seen in Fig. 7.6c, where the initial trajectory has been adapted so as to go through the marked via-points. The learning curves for those variants considered of most interest in Table 7.3 are also shown in Fig. 7.6d.

Table 7.3: Results for the 10-DoF planar arm experiment displaying $(-\log_{10}(-R))$, R being the reward. DR-DMP variants for several reduced dimensions after the indicated number of updates, using 1 or 2 coordination matrices, were tested.

Dimension	**1 update**	**10 updates**	**25 updates**	**50 updates**	**100 updates**	**200 updates**
DR-DMP$_{CMU}(10)$	0.698 ± 0.070	1.244 ± 0.101	1.713 ± 0.102	1.897 ± 0.038	1.934 ± 0.030	1.949 ± 0.026
DR-DMP$_{CMU}(8)$	0.724 ± 0.073	1.265 ± 0.155	1.617 ± 0.135	1.849 ± 0.079	1.905 ± 0.072	1.923 ± 0.075
DR-DMP$_{CMU}(5)$	$\mathbf{0.752 \pm 0.117}$	$\mathbf{1.304 \pm 0.098}$	1.730 ± 0.108	1.910 ± 0.076	1.954 ± 0.063	1.968 ± 0.064
DR-DMP$_{CMU}(2)$	0.677 ± 0.063	1.211 ± 0.093	$\mathbf{1.786 \pm 0.073}$	$\mathbf{1.977 \pm 0.040}$	$\mathbf{1.993 \pm 0.039}$	$\mathbf{1.997 \pm 0.037}$
DR-DMP$_{CMU}(1)$	0.612 ± 0.057	1.161 ± 0.103	1.586 ± 0.071	1.860 ± 0.056	1.931 ± 0.041	1.951 ± 0.042
DR-DMP$_{MCMU}(2, 10)$	$\mathbf{0.738 \pm 0.094}$	$\mathbf{1.304 \pm 0.074}$	1.666 ± 0.159	1.848 ± 0.117	1.893 ± 0.080	1.919 ± 0.054
DR-DMP$_{MCMU}(2, 8)$	0.676 ± 0.123	1.270 ± 0.168	1.681 ± 0.148	1.883 ± 0.058	1.927 ± 0.038	1.939 ± 0.036
DR-DMP$_{MCMU}(2, 5)$	0.687 ± 0.052	1.264 ± 0.113	1.684 ± 0.130	1.897 ± 0.091	1.950 ± 0.056	1.962 ± 0.054
DR-DMP$_{MCMU}(2, 2)$	0.704 ± 0.055	1.258 ± 0.130	$\mathbf{1.749 \pm 0.162}$	$\mathbf{1.976 \pm 0.029}$	$\mathbf{2.000 \pm 0.016}$	$\mathbf{2.006 \pm 0.017}$
DR-DMP$_{MCMU}(2, 1)$	0.579 ± 0.076	1.103 ± 0.125	1.607 ± 0.131	1.885 ± 0.107	1.959 ± 0.055	1.972 ± 0.054
IDR-DMP$_{CMU}$	0.715 ± 0.140	1.195 ± 0.091	1.672 ± 0.121	1.952 ± 0.040	1.997 ± 0.034	2.004 ± 0.030
IDR-DMP$_{MCMU}(2)$	0.656 ± 0.058	1.174 ± 0.158	1.683 ± 0.169	1.937 ± 0.067	$\mathbf{2.013 \pm 0.030}$	$\mathbf{2.019 \pm 0.028}$

In Table 7.3 we can see that:

- Using two coordination matrices yields better results than using one; except for the case of a fixed dimension set to 10, where the coupling matrices would not make sense as they would have full rank.
- Among all the fixed dimensions, the one showing the best results is $r = 2$, which is indeed the dimension of the implicit task in the Cartesian space.
- The variable-dimension iterative method produces the best results.

7-DoF arm circle-drawing experiment

As a second experiment, we kinesthetically taught a real robot - a 7-DoF Barrett's WAM robot - to perform a 3-D circle motion in space.

Initialization and reward function

We stored the real robot data obtained through such kinesthetic teaching and a plot of the end-effector's trajectory together with the closest circle can be seen in Fig. 7.7a.

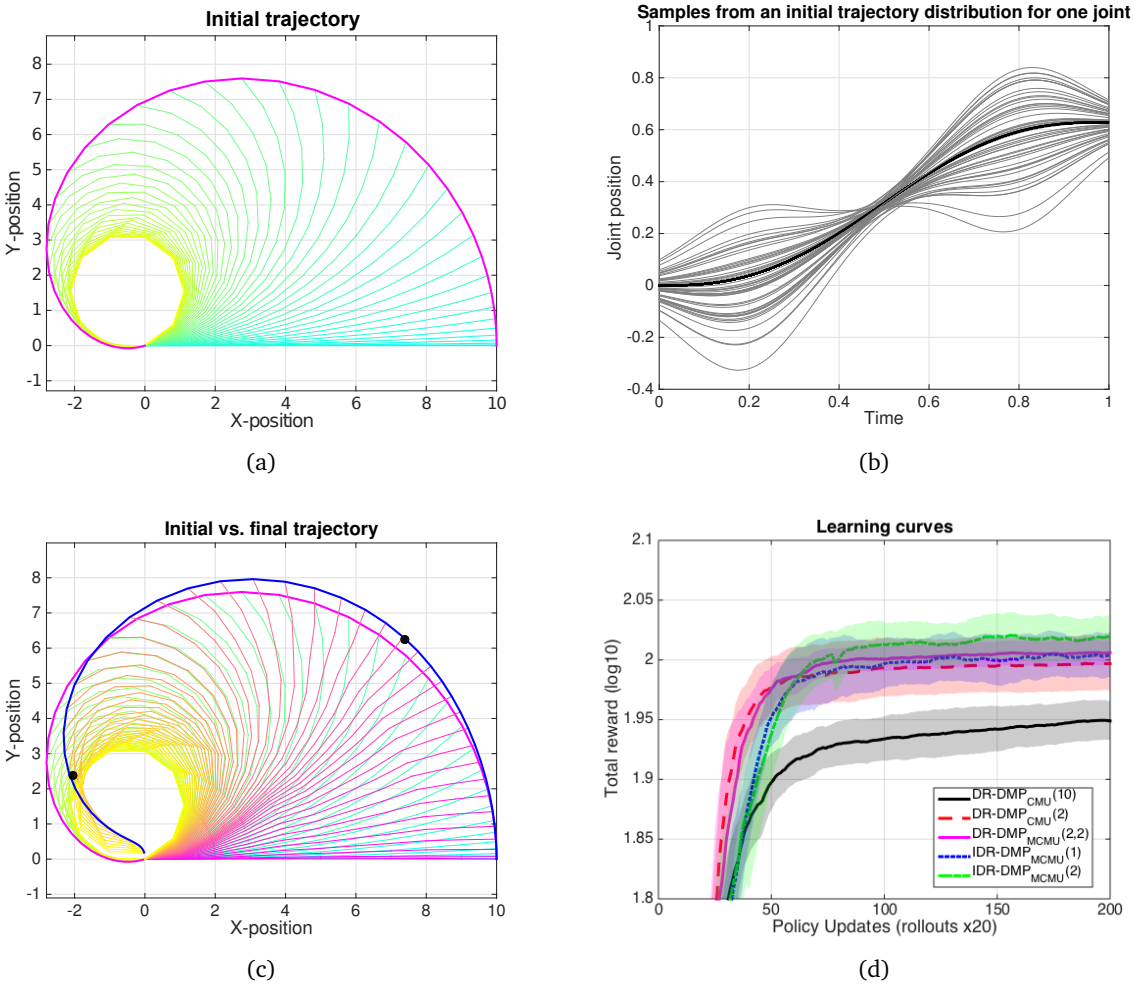
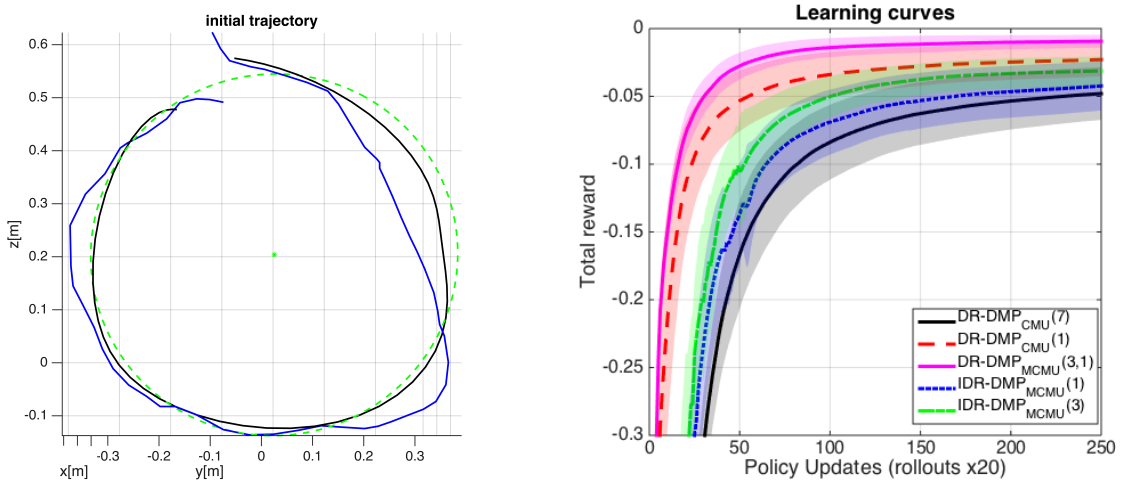


Figure 7.6: 10-DoF planar arm experiment. (a) Joints min-jerk trajectory in the Cartesian XY space of a 10-DoF planar robot arm. The robot links move from the initial position (cyan color) to the end position (yellow), while the end-effector's trajectory is plotted in red. (b) Data generated to initialize the DMP for a single joint. (c) Initial trajectory (magenta) vs. final trajectory (blue) obtained with the IDR-DMP algorithm; via points plotted in black. (d) Learning curves showing mean and 95% confidence interval.



(a) (b)

Figure 7.7: 7-DoF WAM robot experiment. (a) End-effector's trajectory (blue) resulting from a human kinesthetically teaching a WAM robot to track a circle and closest circle in the three-dimensional Cartesian space (green), which is the ideal trajectory; in black, one of the best solutions obtained through PS. (b) Learning curves showing mean and 95% confidence intervals for some of the described methods.

Using again REPS as PS algorithm with $\epsilon_{\text{KL}} = 0.5$, 10 simulated experiments of 250 policy updates consisting of 20 rollouts each were performed, reusing the data of up to the previous 4 epochs. Using the same REPS parameters, we ran the learning experiment with one constant coordination matrix $\mathbf{\Omega}$ and different dimensions $r = 1..7$, namely DR-DMP$^0(r)$. We also ran the experiment updating the coordination matrix after each updated, with a constant dimension, DR-DMP$_{CMU}(r)$. Similarly, we ran the learning experiments with $N_s = 3$ coordination matrices: With constant coordination matrices initialized at the first iteration, DR-DMP$^0_{MCMU}(3, r)$, and updating the coordination matrices at every policy update, DR-DMP$_{MCMU}(3, r)$. We ran the iterative dimensionality reduction with $N_s = 1$ coordination matrix, IDR-DMP$_{CMU}$, and with $N_s = 3$ matrices, IDR-DMP$_{MCMU}(3)$ and last, the EM approach EM DR-DMP(r).

As a reward function, we used:

$$R = -\left(\sum_{t=1}^{N_t} r^t_{\text{circle}} + \alpha \|\ddot{\mathbf{q}}\|^2\right), \tag{7.51}$$

where r_{circle} is the minimum distance between the circle and each trajectory point, $\|\ddot{\mathbf{q}}\|^2$ is the squared norm of the acceleration at each trajectory point, and $\alpha = \frac{1}{5 \cdot 10^6}$ is a constant so as to keep the relative weight of both terms in a way the acceleration represents a value between 10%

Table 7.4: Results for the real data simulated 7-DoF WAM experiment. Rewards are shown for most variants of the DR-DMP methods. See Table 7.1 for a description of the notation used.

Method	**1 update**	**10 updates**	**25 updates**	**50 updates**	**100 updates**	**200 updates**
DR-DMP$^0(7)$	-1.812 ± 0.076	-0.834 ± 0.078	-0.355 ± 0.071	-0.165 ± 0.034	-0.085 ± 0.024	-0.054 ± 0.019
DR-DMP$^0(6)$	-1.801 ± 0.053	-0.836 ± 0.058	-0.359 ± 0.058	-0.169 ± 0.040	-0.093 ± 0.027	-0.060 ± 0.021
DR-DMP$^0(5)$	-1.810 ± 0.047	-0.834 ± 0.058	-0.372 ± 0.048	-0.164 ± 0.037	-0.086 ± 0.028	-0.057 ± 0.023
DR-DMP$^0(4)$	-1.879 ± 0.054	-0.901 ± 0.043	-0.405 ± 0.059	-0.189 ± 0.037	-0.093 ± 0.025	-0.061 ± 0.021
DR-DMP$^0(3)$	-1.523 ± 0.057	-0.765 ± 0.051	-0.340 ± 0.052	-0.162 ± 0.038	-0.096 ± 0.030	-0.072 ± 0.025
DR-DMP$^0(2)$	-1.833 ± 0.065	-0.906 ± 0.074	-0.422 ± 0.063	-0.230 ± 0.043	-0.140 ± 0.029	-0.099 ± 0.024
DR-DMP$^0(1)$	$\mathbf{-0.860 \pm 0.030}$	$\mathbf{-0.447 \pm 0.031}$	$\mathbf{-0.195 \pm 0.031}$	$\mathbf{-0.103 \pm 0.022}$	$\mathbf{-0.068 \pm 0.022}$	$\mathbf{-0.052 \pm 0.021}$
DR-DMP$_{CMU}(7)$	-1.970 ± 0.080	-0.907 ± 0.072	-0.386 ± 0.045	-0.167 ± 0.040	-0.084 ± 0.028	-0.053 ± 0.021
DR-DMP$_{CMU}(6)$	-1.949 ± 0.071	-0.878 ± 0.078	-0.367 ± 0.066	-0.175 ± 0.046	-0.090 ± 0.028	-0.060 ± 0.021
DR-DMP$_{CMU}(5)$	-1.925 ± 0.073	-0.897 ± 0.078	-0.410 ± 0.076	-0.192 ± 0.042	-0.092 ± 0.025	-0.056 ± 0.019
DR-DMP$_{CMU}(4)$	-2.146 ± 0.085	-1.108 ± 0.150	-0.386 ± 0.113	-0.174 ± 0.055	-0.094 ± 0.036	-0.067 ± 0.031
DR-DMP$_{CMU}(3)$	-0.678 ± 0.274	-0.330 ± 0.122	-0.141 ± 0.058	-0.073 ± 0.033	-0.044 ± 0.025	-0.030 ± 0.021
DR-DMP$_{CMU}(2)$	-0.758 ± 0.329	-0.401 ± 0.155	-0.173 ± 0.074	-0.085 ± 0.045	-0.043 ± 0.024	-0.027 ± 0.014
DR-DMP$_{CMU}(1)$	$\mathbf{-0.563 \pm 0.108}$	$\mathbf{-0.216 \pm 0.081}$	$\mathbf{-0.094 \pm 0.036}$	$\mathbf{-0.053 \pm 0.022}$	$\mathbf{-0.034 \pm 0.017}$	$\mathbf{-0.025 \pm 0.013}$
DR-DMP$^0_{MCMU}(3,7)$	-1.880 ± 0.080	-0.854 ± 0.061	-0.384 ± 0.047	-0.196 ± 0.041	-0.107 ± 0.032	-0.070 ± 0.024
DR-DMP$^0_{MCMU}(3,6)$	-1.937 ± 0.064	-0.901 ± 0.098	-0.371 ± 0.063	-0.164 ± 0.041	-0.075 ± 0.024	-0.046 ± 0.018
DR-DMP$^0_{MCMU}(3,5)$	-1.995 ± 0.087	-0.916 ± 0.072	-0.363 ± 0.067	-0.165 ± 0.039	-0.083 ± 0.027	-0.055 ± 0.019
DR-DMP$^0_{MCMU}(3,4)$	-2.169 ± 0.054	-1.108 ± 0.115	-0.355 ± 0.103	-0.154 ± 0.058	-0.081 ± 0.035	-0.053 ± 0.022
DR-DMP$^0_{MCMU}(3,3)$	-0.468 ± 0.016	-0.239 ± 0.020	-0.104 ± 0.016	-0.054 ± 0.011	-0.029 ± 0.006	-0.017 ± 0.004
DR-DMP$^0_{MCMU}(3,2)$	-0.499 ± 0.016	-0.272 ± 0.025	-0.117 ± 0.016	-0.057 ± 0.010	-0.031 ± 0.006	-0.021 ± 0.005
DR-DMP$^0_{MCMU}(3,1)$	$\mathbf{-0.462 \pm 0.033}$	$\mathbf{-0.146 \pm 0.014}$	$\mathbf{-0.061 \pm 0.012}$	$\mathbf{-0.031 \pm 0.008}$	$\mathbf{-0.019 \pm 0.006}$	$\mathbf{-0.012 \pm 0.003}$
DR-DMP$_{MCMU}(3,7)$	-1.970 ± 0.074	-0.872 ± 0.059	-0.390 ± 0.045	-0.181 ± 0.026	-0.086 ± 0.021	-0.050 ± 0.016
DR-DMP$_{MCMU}(3,6)$	-1.981 ± 0.053	-0.929 ± 0.117	-0.376 ± 0.086	-0.181 ± 0.060	-0.084 ± 0.036	-0.050 ± 0.021
DR-DMP$_{MCMU}(3,5)$	-1.951 ± 0.058	-0.851 ± 0.075	-0.320 ± 0.050	-0.133 ± 0.025	-0.068 ± 0.019	-0.045 ± 0.016
DR-DMP$_{MCMU}(3,4)$	-2.165 ± 0.053	-1.090 ± 0.133	-0.322 ± 0.107	-0.142 ± 0.059	-0.075 ± 0.034	-0.051 ± 0.022
DR-DMP$_{MCMU}(3,3)$	-0.456 ± 0.014	-0.244 ± 0.021	-0.112 ± 0.021	-0.057 ± 0.011	-0.030 ± 0.009	-0.017 ± 0.005
DR-DMP$_{MCMU}(3,2)$	-0.500 ± 0.019	-0.285 ± 0.025	-0.134 ± 0.022	-0.070 ± 0.016	-0.037 ± 0.009	-0.022 ± 0.007
DR-DMP$_{MCMU}(3,1)$	$\mathbf{-0.492 \pm 0.032}$	$\mathbf{-0.150 \pm 0.016}$	$\mathbf{-0.062 \pm 0.015}$	$\mathbf{-0.028 \pm 0.010}$	$\mathbf{-0.014 \pm 0.007}$	$\mathbf{-0.010 \pm 0.005}$
IDR-DMP$_{CMU}$	-1.826 ± 0.093	-0.815 ± 0.089	-0.300 ± 0.063	-0.137 ± 0.038	-0.069 ± 0.024	-0.047 ± 0.019
IDR-DMP$_{MCMU}(3)$	-1.955 ± 0.085	-0.807 ± 0.127	-0.240 ± 0.054	-0.105 ± 0.040	-0.050 ± 0.015	-0.033 ± 0.011
EM DR-DMP(6)	-2.288 ± 0.021	-0.884 ± 0.105	-0.180 ± 0.027	-0.086 ± 0.012	-0.051 ± 0.003	-0.043 ± 0.002
EM DR-DMP(5)	-2.274 ± 0.018	-0.975 ± 0.175	-0.217 ± 0.034	-0.094 ± 0.015	-0.056 ± 0.006	-0.043 ± 0.002
EM DR-DMP(4)	-2.363 ± 0.026	-0.926 ± 0.144	-0.194 ± 0.026	-0.100 ± 0.017	-0.059 ± 0.006	-0.044 ± 0.002
EM DR-DMP(3)	-2.050 ± 0.027	-1.075 ± 0.155	-0.231 ± 0.064	-0.103 ± 0.018	-0.063 ± 0.003	-0.049 ± 0.002
EM DR-DMP(2)	-2.271 ± 0.015	-1.417 ± 0.100	-0.335 ± 0.049	-0.203 ± 0.014	-0.145 ± 0.009	-0.091 ± 0.007
EM DR-DMP(1)	-1.111 ± 0.015	-0.653 ± 0.072	-0.283 ± 0.028	-0.196 ± 0.006	-0.182 ± 0.001	-0.177 ± 0.001

and 20% of the cost function.

Results and discussion

The results shown in Table 7.4 have the mean values throughout the 10 experiments, and their confidence intervals with 95% confidence. Figure 7.7b shows the learning curves for some selected methods. Using the standard DMP representation as the benchmark for comparison, with $r = 7$ as fixed dimension (see first row in Table 7.4), we can say that:

- Using $N_s = 3$ coordination matrices yields significantly better results than using only one. $N_s = 3$ with a single dimension results in a final reward of -0.010 ± 0.008, the best

obtained throughout all experiments.

- It is indeed better to use a coordination matrix update with automatic dimensionality reduction than to use the standard DMP representation. Additionally, it provides information on the true underlying dimensionality of the task itself. In the considered case, there is a significant improvement from $r = 4$ to $r = 3$, given that the reward doesn't take orientation into account and, therefore, the task itself lies in the $3-$dimensional Cartesian space. Moreover, the results indicate that a 1-dimensional representation can be enough for the task.

- Fixing the dimension to 1 leads to the best performance results overall, clearly showing that smaller parameter dimensionality yields better learning curves. It is to be expected that, given a 1-dimensional manifold of the Cartesian space, i.e., a trajectory, there exists a 1-dimensional representation of such trajectory. Our approach seems to be approaching such representation, as seen in black in Fig. 7.7a.

- Both DR-DMP$_{CMU}(r)$ and DR-DMP$_{MCMU}(3, r)$ provide a significant improvement over the standard DMP representation, DR-DMP$^0(r)$. This is specially noticeable for $r \leq 3$, where the final reward values are much better. Additionally, the convergence speed is also significantly faster for such dimensions, as the 10 updates column shows.

- The EM DR-DMP(r) shows a very fast convergence to high-reward values for $r = \{4, 5, 6\}$, but the results for a smaller dimension present a premature convergence to less optimal reward values.

14-DoF dual-arm real-robot experiment

As a third experiment, we implemented the same framework on a dual-arm setting consisting of two Barrett's WAM robots, aiming to fold a polo shirt as seen in Fig. 7.8. We placed the robots in a position yielding a good coordination capability according to our previous work in Section 3.2 and then taught the robots the initial motion.

Initialization and reward function

We kinesthetically taught both robots to jointly fold a polo shirt, with a relatively wrong initial attempt as shown in Fig. 7.9a. Then we performed 3 runs consisting of 15 policy updates of 12 rollouts each (totaling 180 real robot dual-arm motion executions for each run), with a reuse of the previous 12 rollouts for the PS update. We ran the standard method and the automatic dimensionality reduction for both $N_s = 1$ and $N_s = 3$. This added up to a total of 540 real robot executions with the dual-arm setup.

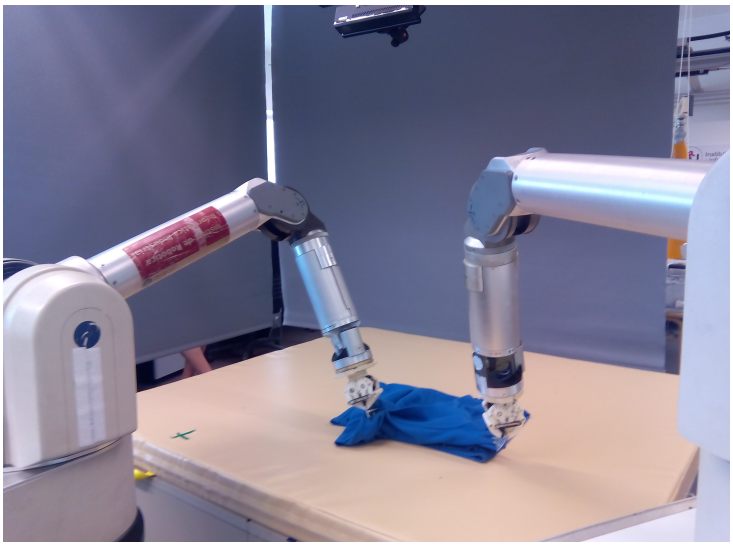


Figure 7.8: Two robotic arms coordinating their motions to fold a polo shirt.

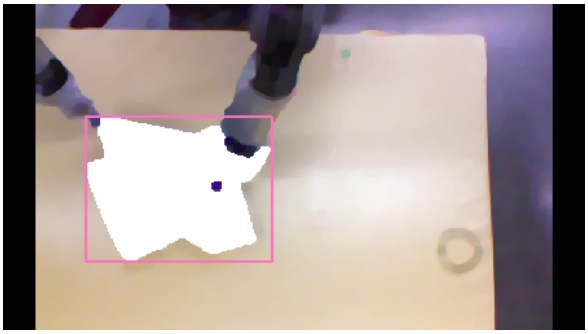
The trajectories were stored in Cartesian coordinates, using 3 variables for position and 3 more for orientation[1], totaling 12 DoF and, with 20 DMP weights per DoF, adding up to 240 DMP parameters. Additionally, the inverse kinematics algorithm in Section 3.1.5 was used to convert Cartesian position trajectories to joint trajectories, which then a computed-torque controller [18] would compliantly track, based on the friction model obtained in Section 4.2.

As reward, we used a function that would penalize large joint accelerations, together with an indicator of how well the polo shirt was folded. To do so, we used a rooftop-placed *Kinect* camera to generate a depth map with which to evaluate the resulting wrinkleness of the polo. Additionally, we checked its rectangleness by color-segmenting the polo on the table and fitting a rectangle with the obtained result (see Fig. 7.9a). Therefore, the reward function used was:

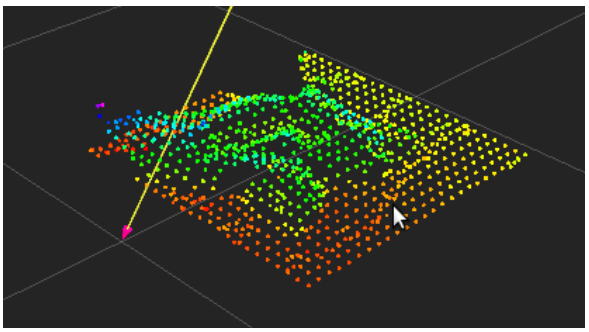
$$R = -R_{\text{acceleration}} - R_{\text{color}} - R_{\text{depth}}, \tag{7.52}$$

where $R_{\text{acceleration}}$ is a term penalizing large acceleration commands at the joint level, R_{color} has a large negative value if the result after the motion does not have a rectangular projection on the table (see Fig. 7.9a) and R_{depth} penalizes the outcome if the polo presents a too wrinkled configuration after the motion (see Fig. 7.9b).

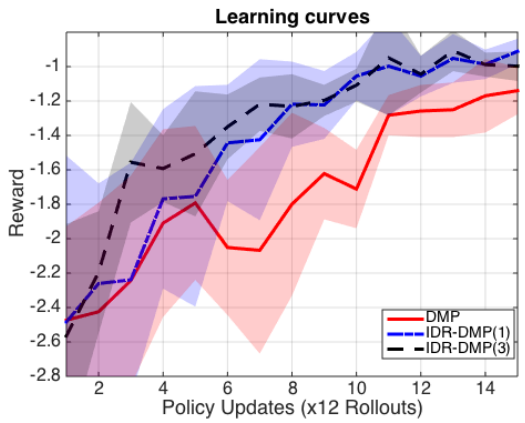
[1] In order to represent orientation as a 3-dimensional state, we used a projection of the quaternion representation $\mathbf{q} = q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}, q_w = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2} \rightarrow (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$. The term $q_w = \cos\frac{\theta}{2}$ can be reconstructed by $\cos\frac{\theta}{2} = \text{cosasin}(\|(q_x, q_y, qz)\|)$. However, such reconstruction looses a sign on the process. To solve such problem, the initial orientation was set to be the identity, and the trajectories' orientations where encoded relative to the initial orientation of the trajectory and, therefore, avoiding such change of sign unless big changes of orientation occur in the trajectory, which was not the case.



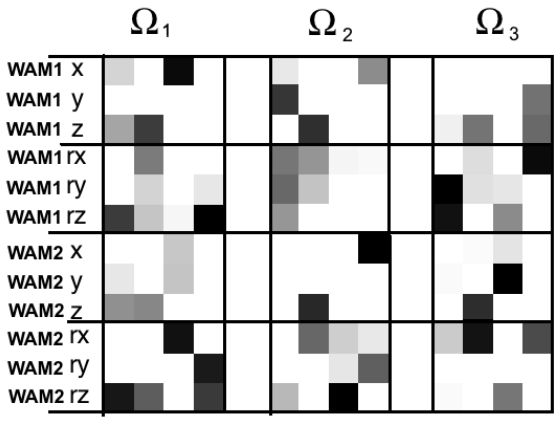
(a)



(b)



(c)



(d)

Figure 7.9: 14-DoF dual-arm real-robot experiment. (a) Color segmentation of the polo shirt after an initial folding attempt. The blue color (as the polo color) was segmented and the number of blue pixels within the smallest rectangle containing it was counted. Then the ratio of blue pixels wrt. the total number of pixels within the rectangle was used for the reward function. (b) Depth image visualization from which the R_{depth} component of the reward function was computed. The mean gradient of the depth was used as a wrinkleness indicator. (c) Learning curves showing mean and standard deviation over the rollouts for each epoch, for the standard setting and two variants of the DR-DMP method. (d) Graphical representation of the synergies obtained for the IDR-DMP(3) method. black areas indicate a higher influence (in absolute value), while white areas represent a small influence.

Results and discussion

Despite the efforts of the authors to reduce the variability of results wrt. environmental uncertainties, a slightest variation of the setup would change the outcome. As an example, we ran the initial attempt with the same DMP parameters (shown in Figs. 7.9a and 7.9b) 20 times with no exploration, yielding a mean and a standard deviation for the vision terms of the reward function of $R_{\text{depth}} = 0.473 \pm 0.029$ and $R_{\text{color}} = 0.799 \pm 0.088$. This uncertainty increased with exploration, resulting in slower learning curves than expected. Nonetheless, the resulting learning curves obtained from the experiments and displayed in Fig. 7.9c show:

- The standard representation of DMPs, with 240 parameters, leads to a long transient period of very small improvement, specially between epochs 4 and 10. This is due to the large number of parameters wrt. the number of rollouts performed.

- The automatic dimensionality reduction with $N_s = 1$ algorithm presents a better and more stable improvement behavior. Ending with a reduced dimension of $r = 6$, reduces the dimensionality in the parameter space down to 120 parameters which, specially in the early stages, allows to keep improving over epochs.

- The automatic dimensionality reduction with $N_s = 3$ algorithm, IDR-DMP(3,12) ending with a dimensionality of $r = 4$, meaning 80 DMP weights - a third of those in the standard method- has a quicker learning at the early epochs, thanks to being able to quickly eliminate unnecessary exploration. After a certain number of iterations, the IDR-DMP(3) algorithm ends with a very similar result to that of IDR-DMP(1).

A video showing some snapshots of this experiment is provided in Appendix B.6. In such video, we can also see the complexity of the task itself as some humans struggle to correctly fold a polo shirt. Our method allows a dual-arm robot to improve its folding skill from an initial faulty demonstration. Using these kinds of RL allows to improve a robotic behavior over an initial existing one.

Additionally, Fig. 7.9d shows a graphical interpretation of the 3 coordination matrices obtained by the IDR-DMP(3) method. Darker areas are more correlated than lighter ones. Knowing that, within each 12×4 matrix, the columns on the left are more relevant, some symmetries can already be seen. While the robot relative positioning seen in Fig. 7.8 does not allow for a perfect match in x and y between the two robots, the z component of both robots is strongly correlated.

Table 7.5: Results for the dual-arm real-robot experiment of folding clothes. The rewards for the three tested methods are shown with the average over rollouts and their standard deviation at each epoch.

method	**1 update**	**2 update**	**5 update**	**10 update**	**15 update**
DR-DMP$^0(12)$	-2.474 ± 0.546	-2.424 ± 0.634	-1.792 ± 0.447	-1.713 ± 0.228	-1.140 ± 0.136
IDR-DMP$_{CMU}$	-2.484 ± 0.966	-2.261 ± 0.582	-1.754 ± 0.641	-1.055 ± 0.141	-0.911 ± 0.074
IDR-DMP$_{MCMU}(3)$	-2.565 ± 0.650	-2.195 ± 0.354	-1.507 ± 0.364	-1.108 ± 0.095	-0.996 ± 0.086

7.2.4 Conclusions

Throughout this section, we proposed different ways to perform task-oriented linear dimensionality reduction of DMPs characterizations of motion. Such approaches help reduce the parameter dimensionality, allowing for faster convergence, while still keeping good optimality conditions.

We presented an algorithm to update the linear dimensionality reduction projection matrix with a task-related weighting in Section 7.2.1, so that it better adapts to the directions expected to provide the most gain in performance in the following steps. We showed how to remove unnecessary parameters from the trajectory representation, by discovering couplings between the robot's degrees of freedom and removing the redundant ones. Both these approaches were combined and extended to use several projection matrices, yielding improved behavior.

The results of the experiments performed (the fully-simulated, the hybrid real-data simulated, and the dual-arm real-robot experiment) clearly show the advantages of using dimensionality reduction for improving PS results with DMP motion characterizations.

Additional Expectation-Maximization (EM) derivations were also tested for such linear dimensionality reduction, but those showed a more greedy behavior in the policy updates, resulting in premature convergence. In fact, EM approaches suffer from such premature convergence, as some literature [146] points out and tries to adapt such covariance in a less greedy manner. We plan to further study and develop such approach in the future.

Moreover, arbitrary deciding a reward function might generate unexpected solutions. To that matter, inverse reinforcement learning can infer a reward function for a certain task under some expertise assumptions on the demonstrated motions to the robot [147], and future developments of this work also go in that direction. Last, the analysis of the obtained coordination matrices $\mathbf{\Omega}$ for the dual-arm experiment indicated an existing symmetry between the two robotic arms. While this symmetry was not in all coordinates, due to the relative positioning of both arms, it points out the possibility of using such symmetries in the future for improving a learning behavior.

7.3 Summary

Using MPs as motion characterization for robot learning leads to a kind of exploration vs. exploitation trade-off (i.e., learning speed vs. solution quality). Such trade-off meaning that a good fitting of the initial trajectory yields too many parameters to effectively perform PS to improve the robot behavior, while too few parameters allows for faster improvements, but limit the optimality of the solution found after learning.

In general, when fitting a robot motion with a certain parametrized movement primitive, it is common to have some overfitting that might result in meaningless exploration when learning. Such overfitting might be useful to have a wider range of exploration in early stages, but quickly eliminating it shows a significant improvement in the learning process of robotic skills.

This chapter showed how complex real robot manipulator tasks, encoded by MPs (either ProMPs in Section 7.1 or DMPs in Section 7.2) can have their number of parameters significantly reduced, understanding MP parameters as those used for exploration. To this endeavor, orienting the DR with the reward of the samples obtained is a crucial element, allowing for a latent projection that encodes that information related to the task only. The information provided by such projection matrices alone is also a qualitative representation of a motion, indicating which DoF synergies are the most actuated.

8 Conclusions

In the twenty-first century,
the robot will take the place which slave labor occupied in ancient civilization.
Nikola Tesla, 1935.

This thesis has focused on a complete framework for learning motion tasks in mostly unmodelled environments with robotic arms. We devised strategies to compliantly learn such tasks both in the joint space and in the robot's operational - or Cartesian- space, as well as to obtain coordination schemes for the robot's DoF for a certain task. This coordinated motion is represented in a compact characterization through linear dimensionality reduction on movement primitives' parameters. Moreover, such compact representation speeds up the learning speed of a task, thanks to the elimination of irrelevant directions of exploration.

8.1 Summary

In this thesis, we have contributed to apparently different but strongly related topics regarding robotics: kinematics, control, motion characterization and learning. None of these elements can effectively work without the others in real-robot learning applications, therefore these topics need to be combined to obtain the best-performing setup.

The contributions of the first part of this thesis can be summed up as follows:

1. We exhaustively reviewed, analyzed and improved some alternatives of Closed-Loop Inverse Kinematics (CLIK) algorithms. CLIK methods in literature were found not to be robust enough regarding convergence and numerical stability, as we pointed out in Chapter 3. Subsequently, we analyzed how robust these methods were in those terms, which are often not considered in literature, and proposed a more robust solution that ensures a numerical stability and better convergence overall. We used these IK approaches to then

elaborate on the bimanual robot configuration, a rather unexplored topic but soon to gain relevance due to bimanual robots becoming more popular.

2. We modeled robot's dynamics in two ways in Chapter 4: using Gaussian fitting methods and building a global friction model. Such models allowed to compliantly move the robot and simultaneously estimate the external forces acting on the robot. We also applied such control algorithms to reproduce trajectories in both the joint space and the Cartesian space, with the help of the IK algorithms obtained in Chapter 3. We used this compliant control together with Movement Primitives (MP) and direct Policy Search (PS) to learn complex tasks like putting a scarf around a mannequin's head or folding a polo shirt. While we added simple camera information to these learning cases, the external force estimation also provided valuable information in terms of both safety and punishing those trajectories in which a large interaction force was detected.

While such framework was positively working with standard MP representations and PS algorithms, other factors could be making the learning process more difficult. In the second part, we focused on:

3. Devising a variant or generalization of a popular PS algorithm that would not get stuck in-between optimal solutions. Overall, PS algorithms applied to robotics start from a movement initialization and locally explore around such provided initial motion. However, when more than one (sub)optimal solution exists, those can be interfering with each other, resulting in a slower convergence of the learning process, subsequently resulting in more samples needed. In the case of human interaction or scenarios with deformable objects, samples must be in a real-robot scenario and have a high cost in terms of physical time. Generalizing REPS to an algorithm that can push the solution away from bad samples, as well as choosing an adequate solution to converge to, helps to improve robot learning performance in such situations. We first tested our proposed approach in a simulated Reinforcement Learning (RL) setting and found that DREPS considerably speeds up the learning process, especially during the early optimization steps and in cases where other approaches get trapped in between several alternative maxima. Further experiments in which a real robot had to learn a task with a multi-modal reward function confirm the advantages of our proposed approach with respect to REPS.

4. Finally, the curse of dimensionality has been defined in many fields. Reinforcement learning, specially PS, use many parameters to optimize. Such number of parameters is usually larger than the number of real robot sample motions we can actually perform. Therefore, not only is the curse of dimensionality affecting the outcome of sampling high dimensional

spaces, but also the fact that if a PS algorithm uses much more parameters than samples to update its policy, the result will be a very greedy learning behavior: The resulting exploration covariance - regularized with a small value in order to keep it full rank - will only explore the parameter subspace that is already known. This fact is actually critical as, given 100 parameters updated with just 20 samples, the resulting exploration matrix will only explore in a subspace of dimension 20 of the parameter space, leaving out all the possibly better solutions that could be found in the higher-dimensional full manifold. While filtering of the covariance matrix can be performed, the algorithms themselves face the dilemma of diluting the full-rank initial exploration with the only obtained samples or have a significantly high regularizing term for exploration that will actually add noise and slow the convergence of the learning. This problem can actually be overcome when using Bernstein's concept of muscle synergy. In the case of robot applications, it will be joint synergy, obtained through linear dimensionality reduction in Chapter 7. Obtaining the DoF's couplings and working in a joint/Cartesian subspace will be less harmful for the task exploration, as the results in Chapter 7 show.

Overall, we provided a complete and vertical framework including kinematics, control, motion characterization and learning, in a manner where coordination between DoF of the robot - either joints or Cartesian space dimensions - is simultaneously learned with the objective task. The reduced number of parameters then helps improve the learning speed. Moreover, all the developed algorithms and applications were implemented on real robots, specially one or two coordinated Barrett WAM arms, and a collection of results and experiments can be seen in Appendix B.

8.2 Future work

In this thesis, we have contributed to build and connect the necessary elements for a model-free reinforcement learning framework with efficient usage of samples in a compliant real-robot environment. Nevertheless, several of the elements built through this thesis can be still further developed or open new directions of research:

- **Kinematics**: This thesis focused on CLIK methods for solving the inverse kinematics of redundant serial robots. While the two enhancements presented in Chapter 3 have been proven to improve their performance, computational speed can still be a limiting factor, specially for the continuous pseudoinverse used in Section 3.1.4. Such computation needs to be optimized or approximated to a faster-to-compute but equivalent one. Moreover, the results in Table 3.5 and Fig. 3.5 point in the direction of needing to combine such inverse

kinematics algorithms with planners, so as to avoid dead-ends in trajectories caused by the unavailability of certain robot assembly modes due to joint limits.

- **Learning motion and forces**: We implemented feed-forward controllers that allow for compliant manipulation, together with disturbance observers for sensor-less robots. We think this approach can be developed further in order to obtain variable-stiffness controllers, using variable gains throughout motion. Other works [148] learn trajectories in both the kinematic and dynamic domain. The integration of such approaches with disturbance observers can lead to similar performance without the need of force sensors.

- **Dimensionality Reduction**: We applied linear DR to movement primitives and obtained the linear projections through principal component analysis and expectation-maximization approaches, for DMPs and ProMPs. Our future aim in this direction is to tackle this problem with other DR techniques - not necessarily linear - as well as to limit the premature convergence experienced with EM approaches for DMPs. Other DR techniques include Factor Analysis, Kernel-based methods, and mixture models for mapping the latent spaces to the full joint space. CMA-ES [135] can also be used to prevent the premature convergence due to covariance shrinkage, to then apply DR techniques on the resulting covariance matrix. However, the natural application of such DR method combined with CMA-ES would require to further evaluate the effects of performing DR in the parameter space, rather in the joint space. Such case will also be considered in the future.

- **Negative samples**: We plan to extend our work in Chapter 6, finding other methods for setting a negative effect on the policy update regarding low-performing samples. Our current approach sets bounds on how close the new policy can be from a cluster of low-performing samples, but more continuous approaches can also be explored.

- **Contextualization**: We strongly think that the direction to go in the future is *adaptability* and *unification*, aiming at better adapting movement primitives with context variables, namely *contextual adaptability*. While some PS approaches have already been adapted to consider context variables [149], movement primitives cannot be as easily adapted to changing situations. Therefore, a better reparametrization of MPs wrt. context variables needs to be studied. Furthermore, it may be the case that the demonstrations taught to the robot in order to initialize a MP are not fully necessary, meaning that motion can be synthetically initialized and then refined through experience.

- **Learning planning and motion**: Reinforcement learning of robotic skills is nowadays still divided into three main areas: perception, planning and motion. While perception is usually linked to the others as valuable information for decision-making or action evaluation,

linking the learning of planning and motion is still rather unexplored. Our belief is that a full learning integration would simultaneously learn which action (motion) to perform, execute it, evaluate it, and then improve it through policy search, while at the same time the motion-selecting planner also learns a better policy as a planner.

8.3 Epilogue

Robotics, as a science field, together with reinforcement learning, have been receiving a lot of attention over the last decades. This is still growing, thanks to the development of hardware that leads to better platforms, with more capabilities in the kinematic, dynamic and computational domains. Up to an extent where certain circles on the society are starting to see robots as a threat for their job positions, fearing to be replaced by a robot in the mid-term future, as N. Tesla actually predicted 80 years ago. However, hardware is developing at a faster pace than software or AI is. Robots are still far from being capable of doing a wide variety of tasks humans can easily perform. The field of reinforcement learning is, therefore, in need of strong development in order to allow robots to do human tasks and actually releasing human of physical work, including dangerous, poorly ergonomic, repetitive or hazardous labour. While latest trends in AI are also using *deep learning* to solve several complex problems, the amount of data in real environments is too limited to apply those deep learning approaches. As shown in our work [23] [133], few samples can suffice to learn simple tasks from scratch, but there's still a long way to make robots capable of correctly adapting motion and compliance to any new situation.

A
List of Publications

In this section, the reader can find the complete list of accepted and submitted publications since the beginning of the PhD:

Journals

1. A. Colomé and C. Torras. "Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements". *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 2, pp. 944-955, 2015.

2. A. Colomé and C. Torras. "Dual REPS: A Generaliztion of Relative Entropy Policy Search Exploiting Bad Experiences", *IEEE Transactions on Robotics*, vol 33, no. 4, 2017.

3. A. Colomé and C. Torras. "Dimensionality Reduction for Dynamic Movement Primitives and Application to Bimanual Manipulation of Clothes". *Under review*.

4. A. Jevtic, A. Colomé, G. Alenyà and C. Torras. "Robot Motion Adaptation through User Intervention and Reinforcement Learning", *Under review*.

Conferences

5. A. Colomé and C. Torras. "Redundant inverse kinematics: Experimental comparative review and two enhancements", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5333-5340, 2012.

6. A. Colomé, D. Pardo, G. Alenyà and C. Torras. "External force estimation during compliant robot manipulation", *IEEE International Conference on Robotics and Automation*, pp. 3535-3540, 2013.

7. A. Colomé and C. Torras. "Positioning two redundant arms for cooperative manipulation of objects", *6th International Workshop on Computational Kinematics*, Vol 15 of Mechanisms and Machine Science, pp. 121-129, 2014.

8. F. Husain, A. Colomé, B. Dellen, G. Alenyà and C. Torras. "Realtime tracking and grasping of a moving object from range video", *IEEE International Conference on Robotics and Automation*, pp. 2617-2622, 2014.

9. A. Colomé, G. Neumann, J. Peters and C. Torras. "Dimensionality reduction for probabilistic movement primitives", *IEEE-RAS International Conference on Humanoid Robots*, pp. 794-800, 2014.

10. A. Colomé and C. Torras. "Dimensionality reduction and motion coordination in learning trajectories with dynamic movement primitives", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1414-1420, 2014.

11. A. Colomé, A. Planells and C. Torras. "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments", *IEEE International Conference on Robotics and Automation*, pp. 5649-5654, 2015.

12. A. Jevtic, A. Colomé, G. Alenyà and C. Torras. "User evaluation of an interactive learning framework for single-arm and dual-arm robots", *8th International Conference on Social Robotics*, pp. 52-61, 2016.

Workshops

13. A. Colomé, D. Pardo, G. Alenyà and C. Torras. "External force estimation for textile grasp detection", *IROS Workshop Beyond Robot Grasping: Modern Approaches for Learning Dynamic Manipulation*, 2012.

14. A. Colomé, G. Alenyà and C. Torras. "Handling high parameter dimensionality in reinforcement learning with dynamic motor primitives", *ICRA Workshop on Novel Methods for Learning and Optimization of Control Policies and Trajectories for Robotics*, 2013.

15. A. Jevtic, A. Colomé, G. Alenyà and C. Torras. "Learning Robot Motion through User Intervention and Policy Search". *ICRA Workshop on Nature versus Nurture in Robotics*, 2016.

B
External Resources

In this section, the reader can find a list of supplementary material for different experiments related with this thesis

B.1 Closed-loop inverse kinematics for redundant robots

http://www.iri.upc.edu/groups/perception/#inverseKinematics

B.2 Friction model applications

http://www.iri.upc.edu/groups/perception/#ScarfTask

B.3 Realtime tracking and grasping of a moving object from range video

http://www.iri.upc.edu/groups/perception/#trackGrasp

B.4 Human-guided compliant control

http://www.iri.upc.edu/groups/perception/#adapt
http://www.iri.upc.edu/groups/perception/#adapt2

B.5 DREPS

http://www.iri.upc.edu/groups/perception/#dualReps

B.6 DR for DMPs

http://www.iri.upc.edu/groups/perception/#drdmp

C
Acronyms

Table C.1: Acronym list

Acronym	Name	Acronym	Name
AI	Artificial Intelligence	JW	Jacobian Weighting
BC	Bounding Cone	KL	Kullback-Liebler Divergence
CF	Conditioning Factor	LfD	Learning from Demonstration
CLIK	Closed Loop Inverse Kinematics	LGP	Local Gaussian Process
CMA	Covariance Matrix Adaptation	LWPR	Locally Weighted Projection Regression
CMU	Coordination Matrix Update	MAE	Mean Absolute Error
CN	Condition Number	MCMU	Multiple Coordination Matrix Update
CTC	Computed Torque Control	MDP	Markov Decision Process
CTP	Continuous Task Priority	MF	Manipulability Factor
DF	Distance Factor	MP	Movement Primitives
DMP	Dynamic Movement Primitive	MSE	Mean Squared Error
DoF	Degrees of Freedom	OF	Orientation Factor
DR	Dimensionality Reduction	PCA	Principal Component Analysis
DREPS	Dual Relative Entropy Policy Search	PI2	Policy Improvement with Path Integrals
ED	Error Damping	PPCA	Probabilistic Principal Component Analysis
EFE	External Force Estimation	ProMP	Probabilistic Movement Primitive
EM	Expectation Maximization	PS	Policy Search
FK	Forward Kinematics	REPS	Relative Entropy Policy Search
GMM	Gaussian Mixture Models	RL	Reinforcement Learning
GP	Gradient Projection	SD	Selective Damping
IDM	Inverse Dynamic Model	SDF	Solution Density Factor
IED	Improved Error Damping	SVD	Singular Value Decomposition
IK	Inverse Kinematics	SVF	Singular Value Filtering
JC	Joint Clamping	TA	Taks Augmentation
JD	Jacobian Damping	TCP	Tool Center Point
JF	Jacobian Filtering	TP	Task Priority
JP	Jacobian Pseudoinverse	WAM	Whole Arm Manipulator
JT	Jacobian Transpose	WMLE	Weighted Maximum Likelihood Estimation

D
Glossary

In this Appendix, we define the most relevant variables used throughout this thesis

Table D.1: Relevant variables (Part I)

Variable	Name
$\mathbf{J}$	Geometric Jacobian
$\mathbf{J}^{\dagger}$	Jacobian pseudoinverse
$\mathbf{q} = [q_1, ..., q_m]$, $\Delta\mathbf{q}$	Joint state, and joint state variation
$\mathbf{x}$	Task space variable
$\mathbf{e}$	Positioning error of the robot
$\kappa(\cdot)$	Condition number of a matrix
$FK(\cdot)$	Forward kinematics function
α	Gain in a CLIK algorithm
$\sigma_1, ..., \sigma_n$	Jacobian singular values
m	Number of joints
n	Task space dimension
$\mathbf{u}_c$	Robot control torque
$\mathbf{u}_e$	External torque acting on the robot
$\mathbf{u}_{PD}$	Proportional-derivative torque
$\mathbf{K}_P$	Proportional gain matrix
$\mathbf{K}_D$	Derivative gain matrix
$\mathbf{M}(\mathbf{q})$	Robot joint inertia matrix
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	Coriolis and centripetal forces
$\mathbf{F}_f$	Robot joint friction
$\mathbf{G}(\mathbf{q})$	Robot's gravity forces
$\mathbf{n}(\mathbf{q}, \dot{\mathbf{q}})$	Coriolis, centripetal, friction and gravity forces

Table D.2: Relevant variables (Part II)

Variable	Name
π	Policy
$\boldsymbol{\Theta}$	Policy parameter space
$\boldsymbol{\theta} \in \boldsymbol{\Theta}$	Policy parameters
$\boldsymbol{\tau}$	Trajectory
$\mathbf{y}$	Robot state variable
$\mathbf{x}$	Robot's latent space state variable
$\mathbf{s}$	Contextual variable
$\mathbf{u}$	Action taken by the policy π
$\mathbf{R}$	Reward function
$\mathbf{J}_{\pi_{\boldsymbol{\theta}}}$	Expected reward under policy π parametrized by $\boldsymbol{\theta}$
$\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$	Sample from a normally distributed policy with mean $\boldsymbol{\mu}_\omega$ and covariance $\boldsymbol{\Sigma}_\omega$
λ, η	Weighting temperatures for PI2 and REPS, respectively
N_t	Number of timesteps of a trajectory
N_f	Number of Gaussian kernels used per DoF
N_k	Number of trajectories (rollouts) used per policy update
$\mathbf{y}_g$	DMP goal position
τ, α_z, β_z, α_x	DMP fixed parameters
x	DMP phase variable
$\mathbf{f}(x) = \mathbf{f}_t$	DMP excitation function
$\mathbf{g}(x)$	N_f-dimensional vector of DMP Gaussian kernels
$\boldsymbol{\psi} = \mathbf{I}_d \otimes \mathbf{g}(x)$	$d \times dN_f$ matrix of Gaussian kernels
$\boldsymbol{\Omega}$	Robot's DoF coupling matrix
$\boldsymbol{\Sigma}_y$	ProMP fitting and system noise
$\boldsymbol{\Sigma}_f$	DMP fitting and system noise

Bibliography

[1] N. Bernstein, *The Co-ordination and Regulation of Movements*. Oxford Pergamon Press, 1967.

[2] "Inner Body: Muscular system." Available at `http://www.innerbody.com/image/musfov.html`.

[3] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[5] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1398–1403, 2002.

[6] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *IEEE International Conference on Robotics and Automation*, pp. 853–858, 2010.

[7] S. Pfeiffer and C. Angulo, "Gesture learning and execution in a humanoid robot via dynamic movement primitives," *Pattern Recognition Letters*, vol. 67, Part 1, pp. 100 – 107, 2015.

[8] L. Rozo, P. Jimenez, and C. Torras, "Robot learning from demonstration of force-based tasks with multiple solution trajectories," in *15th International Conference on Advanced Robotics (ICAR)*, pp. 124–129, 2011.

[9] L. Rozo, S. Calinon, D. G. Caldwell, P. Jiménez, and C. Torras, "Learning Physical Collaborative Robot Behaviors From Human Demonstrations," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 513–527, 2016.

[10] S. M. Khansari-Zadeh and A. Billard, "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[11] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Robot Programming by Demonstration*. Springer, 2008.

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

[13] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[14] A. Colomé and C. Torras, "Redundant inverse kinematics: Experimental comparative review and two enhancements," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5333–5340, 2012.

[15] A. Colomé and C. Torras, "Closed-Loop Inverse Kinematics for Redundant Robots: Comparative Assessment and Two Enhancements," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 2, pp. 944–955, 2015.

[16] W. T. Townsend and J. K. Salisbury, *Mechanical Design for Whole-Arm Manipulation*, pp. 153–164. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993.

[17] A. Colomé and C. Torras, "Positioning two redundant arms for cooperative manipulation of objects," in *6th International Workshop on Computational Kinematics*, pp. 121–129, 2013.

[18] A. Colomé, D. Pardo, G. Alenyà, and C. Torras, "External force estimation during compliant robot manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3535–3540, 2013.

[19] A. Colomé, d. Pardo, G. Alenyà, and C. Torras, "External force estimation for textile grasp detection," in *IROS Workshop Beyond Robot Grasping: Modern Approaches for Learning Dynamic Manipulation*, 2012.

[20] F. Husain, A. Colomé, B. Dellen, G. Alenyà, and C. Torras, "Realtime tracking and grasping of a moving object from range video," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2617–2622, 2014.

[21] A. Colomé, A. Planells, and C. Torras, "A friction-model-based framework for Reinforcement Learning of robotic tasks in non-rigid environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5649–5654, 2015.

[22] A. Jevtic, A. Colomé, G. Alenyà, and C. Torras, "Learning Robot Motion through User Intervention and Policy Search," in *ICRA Workshop on Nature versus Nurture in Robotics*, 2016.

[23] A. Jevtic, A. Colomé, B. Dellen, G. Alenyà, and C. Torras, "User evaluation of an interactive learning framework for single-arm and dual-arm robots," in *8th International Conference on Social Robotics*, pp. 52–61, 2016.

[24] A. Colomé, G. Alenyà, and C. Torras, "Handling high parameter dimensionality in reinforcement learning with dynamic motor primitives," in *ICRA Workshop on Novel Methods for Learning and Optimization of Control Policies and Trajectories for Robotics*, 2013.

[25] A. Colomé and C. Torras, "Dimensionality reduction and motion coordination in learning trajectories with Dynamic Movement Primitives," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1414–1420, 2014.

[26] A. Colomé and C. Torras, "Dimensionality reduction for dynamic movement primitives and application to bimanual manipulation of clothes," *under review*, 2017.

[27] A. Colomé, G. Neumann, J. Peters, and C. Torras, "Dimensionality reduction for probabilistic movement primitives," in *IEEE-RAS 14th International Conference on Humanoid Robots (Humanoids)*, pp. 794–800, 2014.

[28] S. Sasaki, "Feasibility studies of kinematics problems in the case of a class of redundant manipulators," *Robotica*, vol. 13, no. 03, pp. 233–241, 1995.

[29] G. K. Singh and J. Claassens, "An analytical solution for the inverse kinematics of a redundant 7dof manipulator with link offsets," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2976–2982, 2010.

[30] X. Ding and C. Fang, "A novel method of motion planning for an anthropomorphic arm based on movement primitives," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 2, pp. 624–636, 2013.

[31] V. Ruiz de Angulo and C. Torras, "Learning inverse kinematics: Reduced sampling through decomposition into virtual robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 38, no. 6, pp. 1571–1577, 2008.

[32] S. Ulbrich, V. Ruiz De Angulo, T. Asfour, C. Torras, and R. Dillmann, "General robot kinematics decomposition without intermediate markers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 4, pp. 620–630, 2012.

[33] S. Chiaverini, G. Oriolo, and I. D. Walker, "Kinematically redundant manipulators," in *HANDBOOK of ROBOTICS*, Springer, 2008.

[34] K.-Y. Kim, H.-S. Song, J.-W. Suh, and J.-J. Lee, "A novel surgical manipulator with workspace-conversion ability for telesurgery," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 200–211, 2013.

[35] J. Funda, R. H. Taylor, B. Eldridge, S. Gomory, and K. G. Gruben, "Constrained cartesian motion control for teleoperated surgical robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 453–465, 1996.

[36] R. Rao, A. Asaithambi, and S. Agrawal, "Inverse kinematic solution of robot manipulators using interval analysis," *Journal of Mechanical Design*, vol. 120, no. 1, pp. 147–150, 1998.

[37] J. M. Porta Pleite, L. Ros Giralt, and F. Thomas Arroyo, "Inverse kinematics by distance matrix completion," in *12th International Workshop on Computational Kinematics*, pp. 1–9, 2005.

[38] V. Ruiz de Angulo and C. Torras, "Self-calibration of a space robot," *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 951–963, 1997.

[39] V. Ruiz de Angulo and C. Torras, "Speeding up the learning of robot kinematics through function decomposition," *IEEE Transactions on Neural Networks*, vol. 16, no. 6, pp. 1504–1512, 2005.

[40] S. F. Assal, K. Watanabe, and K. Izumi, "Neural network-based kinematic inversion of industrial redundant robots using cooperative fuzzy hint for the joint limits avoidance," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 5, pp. 593–603, 2006.

[41] S. Ulbrich, V. Ruiz de Angulo, T. Asfour, C. Torras, and R. Dillmann, "Rapid learning of humanoid body schemas with kinematic bezier maps," in *9th IEEE-RAS International Conference on Humanoid Robots*, pp. 431–438, 2009.

[42] S. Ulbrich, V. Ruiz de Angulo, T. Asfour, C. Torras, and R. Dillmann, "Kinematic bezier maps," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, no. 4, pp. 1215–1230, 2012.

[43] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[44] D. E. Orin and W. W. Schrader, "Efficient computation of the jacobian for robot manipulators," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 66–75, 1984.

[45] M. Tucker and N. D. Perreira, "Generalized inverses for robotic manipulators," *Mechanism and machine theory*, vol. 22, no. 6, pp. 507–514, 1987.

[46] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on man-machine systems*, vol. 10, no. 2, pp. 47–53, 1969.

[47] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *IEEE Conference on Decision and Control*, pp. 1359–1363, 1984.

[48] S. Chiaverini, O. Egeland, and R. Kanestrom, "Achieving user-defined accuracy with damped least-squares inverse kinematics," in *IEEE Fifth International Conference on Advanced Robotics (ICAR)*, pp. 672–677, 1991.

[49] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator," *IEEE Transactions on Control Systems Technology*, vol. 2, no. 2, pp. 123–134, 1994.

[50] B. Siciliano, "A closed-loop inverse kinematic scheme for on-line joint-based robot control," *Robotica*, vol. 8, no. 03, pp. 231–243, 1990.

[51] H. Das, J.-E. Slotine, and T. Sheridan, "Inverse kinematic algorithms for redundant systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 43–48, 1988.

[52] 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, 2009.

[53] Z.-P. Jiang and Y. Wang, "A converse lyapunov theorem for discrete-time systems with disturbances," *Systems & control letters*, vol. 45, no. 1, pp. 49–58, 2002.

[54] P. Falco and C. Natale, "On the stability of closed-loop inverse kinematics algorithms for redundant robots," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 780–784, 2011.

[55] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *journal of graphics, gpu, and game tools*, vol. 10, no. 3, pp. 37–49, 2005.

[56] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 722–728, 1985.

[57] T. Yoshikawa, "Dynamic manipulability of robot manipulators," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1033–1038, 1985.

[58] T. Yoshikawa, "Analysis and control of robot manipulators with redundancy," in *Robotics research: the first international symposium*, pp. 735–747, Mit Press Cambridge, MA, 1984.

[59] A. S. Deo and I. D. Walker, "Minimum effort inverse kinematics for redundant manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 5, pp. 767–775, 1997.

[60] Y. Nakamura, *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[61] J. Duffy, "The fallacy of modern hybrid control theory that is based on orthogonal complements of twist and wrench spaces," *Journal of Robotic Systems*, vol. 139, no. 2, pp. 144–199, 1990.

[62] K. L. Doty, C. Melchiorri, E. M. Schwartz, and C. Bonivento, "Robot manipulability," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 462–468, 1995.

[63] F. Ranjbaran, J. Angeles, and A. Kecskeméthy, "On the kinematic conditioning of robotic manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 3167–3172, 1996.

[64] F. Aghili, "Adaptive control of manipulators forming closed kinematic chain with inaccurate kinematic model," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 5, pp. 1544–1554, 2013.

[65] I.-W. Park, B.-J. Lee, S.-H. Cho, Y.-D. Hong, and J.-H. Kim, "Laser-based kinematic calibration of robot manipulator using differential kinematics," *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 6, pp. 1059–1067, 2012.

[66] C. A. Klein and B. E. Blaho, "Dexterity measures for the design and control of kinematically redundant manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 72–83, 1987.

[67] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, "An estimate for the condition number of a matrix," *SIAM Journal on Numerical Analysis*, vol. 16, no. 2, pp. 368–375, 1979.

[68] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. Dimarogonas, and D. Kragic, "Dual arm manipulation - A survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340–1353, 2012.

[69] S. H. Hyon, J. G. Hale, and G. Cheng, "Full-Body Compliant Human Humanoid Interaction: Balancing in the Presence of Unknown External Forces," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 884–898, 2007.

[70] N. Vahrenkamp, M. Przybylski, T. Asfour, and R. Dillmann, "Bimanual grasp planning," in *IEEE-RAS 11th International Conference on Humanoid Robots (Humanoids)*, pp. 493–499, 2011.

[71] F. Zacharias, D. Leidner, F. Schmidt, C. Borst, and G. Hirzinger, "Exploiting structure in two-armed manipulation tasks for humanoid robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5446–5452, 2010.

[72] C. Ott, O. Eiberger, W. Friedl, B. Bauml, U. Hillenbrand, C. Borst, A. Albu-Schaffer, B. Brunner, H. Hirschmuller, S. Kielhofer, R. Konietschke, M. Suppa, T. Wimbock, F. Zacharias, and G. Hirzinger, "A Humanoid Two-Arm System for Dexterous Manipulation," in *6th IEEE-RAS International Conference on Humanoid Robots*, pp. 276–283, 2006.

[73] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3229–3236, 2007.

[74] W. Burgard, O. Brock, and C. Stachniss, *Safety Evaluation of Physical Human-Robot Interaction via Crash-Testing*, pp. 1–352. MIT Press, 2008.

[75] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2016.

[76] T. Tamei and T. Shibata, "Fast Reinforcement Learning for Three-Dimensional Kinetic Human–Robot Cooperation with an EMG-to-Activation Model," *Advanced Robotics*, vol. 25, no. 5, pp. 563–580, 2011.

[77] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Computed torque control with nonparametric regression models," in *American Control Conference*, pp. 212–217, 2008.

[78] S. Klanke, S. Vijayakumar, and S. Schaal, "A library for Locally Weighted Projection Regression," *Journal of Machine Learning Research*, vol. 9, pp. 623–626, 2008.

[79] M. V. Damme, P. Beyl, B. Vanderborght, V. Grosu, R. V. Ham, I. Vanderniepen, A. Matthys, and D. Lefeber, "Estimating robot end-effector force from noisy actuator torque measurements," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1108–1113, 2011.

[80] A. Alcocer, A. Robertsson, A. Valera, and R. Johansson, "Force estimation and control in robot manipulators," in *Robot control 2003* (J. Sasiadek and I. Duleba, eds.), pp. 31–36, 2004.

[81] G. McLachlan and D. Peel, *Finite Mixture Models*. Wiley, 2000.

[82] L. R. Rabiner, J. H. McClellan, and T. W. Parks, "FIR digital filter design techniques using weighted Chebyshev approximation," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 595–610, 1975.

[83] D. Mitrovic, S. Nagashima, S. Klanke, T. Matsubara, and S. Vijayakumar, "Optimal Feedback Control for anthropomorphic manipulators," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4143–4150, 2010.

[84] O. Kinematics and Dynamics, "Inverse Kinematics with KDL.."

[85] "ROS package repository with Barrett WAM/Hand interface.." Available at `http://code.google.com/p/lis-ros-pkg/wiki/README`.

[86] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.

[87] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[88] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Journal of dynamic systems, measurement, and control*, vol. 108, no. 3, pp. 163–171, 1986.

[89] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[90] S. K. Chan and P. D. Lawrence, "General inverse kinematics with the error damped pseudoinverse," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 834–839, 1988.

[91] T. Sugihara, "Solvability-unconcerned inverse kinematics by the levenberg–marquardt method," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 984–991, 2011.

[92] T. F. Chan and R. V. Dubey, "A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators," *IEEE transactions on Robotics and Automation*, vol. 11, no. 2, pp. 286–292, 1995.

[93] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.

[94] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[95] L. Sciavicco and B. Siciliano, "A solution algorithm to the inverse kinematic problem for redundant manipulators," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, pp. 403–410, 1988.

[96] J. Xiang, C. Zhong, and W. Wei, "General-weighted least-norm control for redundant manipulators," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 660–669, 2010.

[97] H. Zghal, R. Dubey, and J. Euler, "Efficient gradient projection optimization for manipulators with multiple degrees of redundancy," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1006–1011, 1990.

[98] F. Flacco, A. De Luca, and O. Khatib, "Prioritized multi-task motion control of redundant robots under hard joint constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3970–3977, 2012.

[99] D. Raunhardt and R. Boulic, "Progressive clamping," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4414–4419, 2007.

[100] N. Mansard, A. Remazeilles, , and F. Chaumette, "Continuity of varying-feature-set control laws," *IRISA Technical report*, 2009.

[101] N. Mansard, A. Remazeilles, and F. Chaumette, "Continuity of varying-feature-set control laws," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2493–2505, 2009.

[102] E. Saff and A. Kuijlaars, "Distributing many points on the sphere," *Mathematical Intelligencer*, vol. 19, no. 01, pp. 5–11, 1997.

[103] G. Barequet and G. Elber, "Optimal Bounding Cones of Vectors in Three Dimensions," *Inf. Process. Lett.*, vol. 93, no. 2, pp. 83–89, 2005.

[104] E. Guizzo, *DARPA Seeking to Revolutionize Robotic Manipulation*, 2010.

[105] T. Asfour, K. Berns, and R. Dillmann, "The humanoid robot ARMAR: Design and control," in *IEEE/APS International Conference on Humanoid Robots*, pp. 7–8, 2000.

[106] T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann, "ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pp. 169–175, 2006.

[107] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano, "Global task space manipulability ellipsoids for multiple-arm systems," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 5, pp. 678–685, 1991.

[108] A. Bicchi and D. Prattichizzo, "Manipulability of cooperating robots with unactuated joints and closed-chain mechanisms," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 4, pp. 336–345, 2000.

[109] A. Albu-Schaeffer, O. Eiberger, M. Grebenstein, S. Haddadin, C. Ott, T. Wimboeck, S. Wolf, and G. Hirzinger, "Soft robotics: From torque feedback controlled lightweight robots to intrinsically compliant systems," *IEEE Robotics and Automation Magazine*, vol. 15, no. 3, pp. 20–30, 2008.

[110] P. J. Hacksel and S. E. Salcudean, "Estimation of Environment Forces and Rigid-Body Velocities Using Observers," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 931–936, 1994.

[111] C.-S. Liu and H. Peng, "Inverse-Dynamics Based State and Disturbance Observers for Linear Time-Invariant Systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 124, no. 3, pp. 375–381, 2002.

[112] L. G. W.H. Chen, "Analysis of Disturbance Observer Based Control for Nonlinear Systems under Disturbances with Bounded Variation," in *International Conference on Control*, pp. 1–5, 2004.

[113] W.-H. Chen, "Disturbance observer based control for nonlinear systems," *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 4, pp. 706–710, 2004.

[114] D. Nguyen-Tuong and J. Peters, "Learning Robot Dynamics for Computed Torque Control Using Local Gaussian Processes Regression," in *ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, pp. 59–64, 2008.

[115] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *29th International Conference on Machine Learning*, vol. abs/1206.4621, 2012.

[116] F. Martí, "C++ Library to check if a scarf is well placed on a mannequin." Available at `https://github.com/FelipMarti`.

[117] J. Kober, *Learning Motor Skills: From Algorithms to Robot Experiments*. PhD thesis, TU Darmstadt, 2012.

[118] J. Peters and S. Schaal, ""Reinforcement learning of motor skills with policy gradients"," *Neural Networks*, vol. 21, no. 4, pp. 682 – 697, 2008.

[119] J. Peters and S. Schaal, "Policy Gradient Methods for Robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2219–2225, 2006.

[120] S. Kullback and R. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, 1951.

[121] J. Peters, K. Mülling, and Y. Altün, "Relative Entropy Policy Search," in *AAAI Conference on Artificial Intelligence*, pp. 1607–1612, 2010.

[122] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical relative entropy policy search," *Journal of Machine Learning Research*, vol. 17, no. 93, pp. 1–50, 2016.

[123] G. Neumann, "Variational Inference for Policy Search in changing situations," in *International Conference on Machine Learning*, pp. 817–824, 2011.

[124] E. Theodorou, J. Buchli, and S. Schaal, "A Generalized Path Integral Control Approach to Reinforcement Learning," *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[125] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[126] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1398–1403, 2002.

[127] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 2616–2624, 2013.

[128] A. Lazaric and M. Ghavamzadeh, "Bayesian Multi-Task Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, pp. 599–606, 2010.

[129] A. Colomé and C. Torras, "Dual reps: A generalization of relative entropy policy search exploiting bad experiences," *IEEE Transactions on Robotics*, vol. 33, no. 4, 2017.

[130] V. Gómez, H. J. Kappen, J. Peters, and G. Neumann, "Policy search for path integral control," in *European Conference in Machine Learning and Knowledge Discovery in Databases (ECML)*, pp. 482–497, 2014.

[131] S. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 2006.

[132] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for k-means clustering," *Pattern Recognition Letters*, vol. 25, no. 11, pp. 1293 – 1302, 2004.

[133] A. Jevtic, A. Colomé, G. Alenyà, and C. Torras, "Robot Motion Adaptation through User Intervention and Reinforcement Learning," in *Under review*, 2017.

[134] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning Movement Primitives," in *11th International Symposium on Robotics Research*, pp. 561–572, 2005.

[135] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*, pp. 75–102. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[136] A. Abdolmaleki, N. Lau, L. P. Reis, and G. Neumann, "Regularized covariance estimation for weighted maximum likelihood policy search methods," in *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 154–159, 2015.

[137] R. Vinjamuri, M. Sun, C. C. Chang, H. N. Lee, R. J. Sclabassi, and Z. H. Mao, "Dimensionality Reduction in Control and Coordination of the Human Hand," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 2, pp. 284–295, 2010.

[138] K. S. Luck, G. Neumann, E. Berger, J. Peters, and H. B. Amor, "Latent space policy search for robotics," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1434–1440, 2014.

[139] H. B. Amor, O. Kroemer, U. Hillenbrand, G. Neumann, and J. Peters, "Generalization of human grasping for multi-fingered robot hands," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2043–2050, 2012.

[140] M. Toussaint, *Lecture Notes: Gaussian identities*. Available at `http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gaussians.pdf`.

[141] J. Duchi, *Properties of the Trace and Matrix Derivatives*. Available at `https://web.stanford.edu/~jduchi/projects/matrix_prop.pdf`.

[142] N. Shafii, A. Abdolmaleki, R. Ferreira, N. Lau, and L. P. Reis, "Omnidirectional Walking and Active Balance for Soccer Humanoid Robot," in *16th Portuguese Conference on Artificial Intelligence (EPIA)*, pp. 283–294, 2013.

[143] D. Pardo, *Learning rest-to-rest Motor Coordination in Articulated Mobile Robots*. PhD thesis, Universitat Politècnica de Catalunya, 2009.

[144] R. A. Téllez, C. Angulo, and D. E. Pardo, "Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture," in *International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT)*, pp. 5–19, 2006.

[145] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with EM-based Reinforcement Learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3232–3237, 2010.

[146] P. A. N. Bosman, J. Grahl, and D. Thierens, "Enhancing the Performance of Maximum–Likelihood Gaussian EDAs Using Anticipated Mean Shift," in *International Conference Parallel Problem Solving from Nature (PPSN)*, pp. 133–143, 2008.

[147] S. Zhifei and E. Meng Joo, "A survey of inverse reinforcement learning techniques," *International Journal of Intelligent Computing and Cybernetics*, vol. 5, no. 3, pp. 293–311, 2012.

[148] L. Rozo, J. Silvério, S. Calinon, and D. G. Caldwell, "Learning controllers for reactive and proactive behaviors in human-robot collaboration," *Frontiers in Robotics and AI*, vol. 3, no. 30, pp. 1–11, 2016.

[149] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-efficient Generalization of Robot Skills with Contextual Policy Search," in *AAAI Conference on Artificial Intelligence*, pp. 1401–1407, 2013.