Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik

Human-centered Assistive Robotics (HCR)

Robotic Tasks Acquisition via Human Guidance: Representation, Learning and Execution

Matteo Saveriano

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.) genehmigten Dissertation.

Vorsitzender:	Prof. Gordon Cheng, Ph.D.
Prüfer der Dissertation:	
	1. Prof. Dongheui Lee, Ph.D.
	2. Prof. Alberto Finzi, Ph.D. University of Naples / Italy

Die Dissertation wurde am 10.10.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 02.12.2017 angenommen.

To my parents and my beloved Francesca

Acknowledgment

My passion for science started when I was just a child and constantly grew up during my life. I was always aware about my passion for scientific matters, but, looking at the child I was some years ago, I could never have imagined that I will finally work in robotics. During my childhood and adolescence, I was more fascinated from natural phenomena, and the laws behind them, than from electronics and machines. I fallen in love with robotics during my university studies, thanks to the lectures of prof. Bruno Siciliano. I will always be grateful to prof. Siciliano for his amazing lectures, for his humor, and for his passion for robotics.

Since 2012 I joined the Human-Centered Assistive Robotics group (HCR) at Technical University of Munich, where I had the possibility to conduct amazing research in robotics. I will always be grateful to my advisor Prof. Dr. Dongheui Lee for this great opportunity. She hired me when I was a fresh automatic control graduate and helped me in every aspect of my research. She showed me a novel and exiting research field, where machine learning and control theory are combined together and applied to robotics. Our discussions, her criticisms, and the freedom she gave me to explore new areas significantly contributed to develop my scientifically rigorous mindset. Prof. Lee gave me also the opportunity to work on two amazing research projects, namely the SAPHARI (FP7 ICT-287513) and the ROLITOS (9.01) projects. Thanks to the European community, the German Research Foundation (DFG), and the International Graduate School of Science and Engineering for supporting my research activities.

The HCR team is not big but really heterogeneous. For several mounts I was the only European person in the group. I would like to thank my colleagues Affan Pervez and Shile Li for helping in the Machine Learning in Robotics course, for which we are the teaching assistant. Thanks Karna Potwar for your jokes and for the discussions about Italian and British Soccer Leagues. Sangik An was always an important and mathematically sound opinion to have. Dr. Pietro Falco's mentorship was of fundamental importance in a very delicate moment of my PhD. Thanks Pietro for the never ending discussions on control theory and machine learning, and for the fruitful cooperation. Finally, thanks to Susanne Schneider for helping with TUM bureaucracy and formalities.

During my PhD years I had the chance to spend three mounts abroad. I decided to visit the Prisma Lab at University of Naples, hosted by Prof. Bruno Siciliano and Prof. Alberto Finzi. At a first sight, this choice seems a bit funny. Why going back to the university were I studied instead of visiting a novel place? Well, I conducted a significant amount of research in cooperation with Prof. Finzi and with my friend and colleague Dr. Riccardo Caccavale. Hence, I preferred to continue my research activity instead of visiting a fancy place.

Living in Munich gave me also the opportunity to meet new people and friends. Thanks Gianluca Garofalo, Cristiano Moschini, Daniele Salvigni, Fabio Bracci, Catalina Daniela Istrate, Ksenia Klionovska, Ingrid Cola Cola, Mohsen Koboli, Cristian Axenie, Lorenzo Santoro, and Umberto Scarcia for the great parties in Munich, and thanks to my old friends Vincenzo Nigro, Generoso De Biase, and Claudio Basile for the great parties back home. I was missing you for such a long time, but luckily we are still friends Fabio Carbone and Sonia Stornaioulo. I cannot forget my "babe" Francesca Mele, a light in the darker night of my life. Special thanks to my parents, to my brother Danilo and my sister Ilenia. I would not be here without your infinite support.

Munich, September 2017 Matteo Saveriano

Contents

Abstract

Robotic researchers aim at realizing a general purpose automaton capable of executing a variety of activities. Such a machine will significantly improve our life, by substituting or helping us in heavy, dangerous, or tedious tasks. Imitating the behavior of the most general automaton, which is the human being, is the key for a wide diffusion of robotic assistants in our daily life. By imitating the humans, in fact, the robotic assistant can continuously acquire novel skills, starting from basic movements to complex tasks requiring the manipulation of different objects. Moreover, by observing the way humans interact and cooperate, the robotic assistant can learn how a known task is executed autonomously or in cooperation with human or robotic partners. The goal of this work is to permit an intuitive skill transfer from a human teacher to a robot learner, and an autonomous or cooperative execution of learned tasks in dynamic environments.

In order to achieve this goal, this thesis presents a framework that permits intuitive learning, compact representation, and flexible execution of robotic tasks. Task demonstrations are used to intuitively transfer novel skills to the robot. Task demonstrations are collected via human imitation and robot physical guidance, which are realized through customized controllers. Collected demonstrations are then automatically segmented into basic motion units and used to represent the task knowledge. A learned task is represented at two different levels. At the lower level, each segmented motion unit is compactly encoded into a stable dynamical system. The stable system is used at run time to generate the robot's trajectory. At the higher level, the task is represented in a symbolic manner. The symbolic representation is also learned from demonstration and used to generate the task plan that the robot has to execute. The task execution is orchestrated by an attentional supervisory system. Inspired by the control mechanisms in human beings, the attentional system exploits contextual information to select the most emphasized robot's behavior and generate the task plan. This mechanism is driven by the learned symbolic representation and the current context, allowing a rapid adaptation of the task plan to different scenarios. During the task execution the robot may accidentally hit unforeseen objects in the scene, especially in dynamic environments. To prevent this situation, the robot's trajectory is online reshaped in order to obtain collision-free paths. At the same time, dynamical system properties are exploited to guarantee a correct task execution.

Presented methodologies are evaluated with simulations and experiments on real robots, showing promising results in terms of intuitive task acquisition and flexible execution. In particular, experiments show that the robot is able to rapidly acquire complex tasks, like preparing a certain receipt, and to successfully execute a learned skill autonomously or in cooperation with a human co-worker.

Zusammenfassung

Roboterforscher zielen darauf ab, einen Allzweckautomaten zu realisieren, der in der Lage ist, eine Vielzahl von Aktivitäten auszuführen. Eine solche Maschine kann unser Leben erheblich verbessern, indem sie uns bei schweren, gefäoder langwierigen Aufgaben unterstützt. Damit Roboter Assistenten eine große Vorbereitung in unserem täglichen Leben finden, müssen sie das Verhalten des vielseitigen Automaten überhaupt, nämlich des Menschen, möglichst gut nachempfinden. Durch die Nachahmung des Menschen kann der Roboter Assistent kontinuierlich neue Fähigkeiten erwerben, angefangen von grundlegenden Bewegungen bis hin zu komplexen Aufgaben, die Manipulation verschiedener Objekte erfordern. Darüber hinaus kann der Roboterassistent durch die Beobachtung der Art und Weise, wie Menschen interagieren und zusammenarbeiten, lernen, wie eine bekannte Aufgabe autonom oder in Zusammenarbeit mit menschlichen oder Roboterpartnern durchgeführt wird. Diese Arbeit beschäftigt sich damit, wie Fähigkeiten intuitiv von einem Lehrer zu einem Roboter Lehrenden übertragen und wie die gelernten Aufgaben autonom oder kooperativ in dynamischen Umgebungen ausgeführt werden können.

Zur Erreichung dieses Ziels stellt diese Arbeit einen Rahmen vor, der ein intuitives Lernen, eine kompakte Darstellung und eine flexible Ausführung von Roboteraufgaben ermöglicht. Durch Vorführung von Aufgaben werden intuitiv neue Fähigkeiten an den Roboter übertragen. Task-Demonstrationen werden über menschliche Nachahmung und Roboter-physische Führung gesammelt, die durch kundenspezifische Steuerungen realisiert werden. Die gesammelte Demonstrationen werden dann automatisch in Grundbewegungseinheiten segmentiert und zur Darstellung des Aufgabenwissens verwendet. Eine gelehrte Aufgabe besteht aus zwei Ebenen. Auf unterer Ebene werden Bewegungseinheiten als stabile dynamische Bewegungsmodelle beschreiben. Diese werden zur Laufzeit zur Erzeugung der Trajektorie des Roboters verwendet. Auf oberer Ebene wird die Aufgabe symbolisch dargestellt. Die symbolische Darstellung wird verwendet, um den Aufgabenplan zu generieren, den der Roboter ausführen muss. Die Aufgabenstellung wird durch ein Aufsichtssystem orchestriert. Inspiriert von den natürlichen Kontrollmechanismen eines Menschen nutzt das aufmerksame System kontextbezogene Informationen, um das Verhalten des Roboters auszuwählen und den Aufgabenplan zu generieren. Dieser Mechanismus wird durch die gelernte symbolische Darstellung und den aktuellen Kontext gesteuert, was eine schnelle Anpassung des Aufgabenstellers an verschiedene Szenarien ermöglicht. Während der Ausführung einer Aufgabe kann der Roboter auf unvorhergesehene Objekte in der Szene stoßen, vor allem in dynamischen Umgebungen. Um dies zu verhindern und kollisionsfreie Wege zu erhalten, wird die Trajektorie des Roboters in Echtzeit angepasst. Gleichzeitig werden dynamische Systemeigenschaften ausgenutzt, um eine korrekte Aufgabenausführung zu gewährleisten.

Die vorgestellten Methoden werden mittels Simulationen und Experimenten an realen Robotern ausgewertet. Die Ergebnisse sind in Bezug auf intuitive Aufgabenstellung und flexible Ausführung vielversprechend. Insbesondere zeigen die Experimente, dass der Roboter in der Lage ist, schnell komplexe Aufgaben, wie die Vorbereitung einer bestimmten Quittung zu erlernen und diese daraufhin selbstständig oder in Zusammenarbeit mit einem menschlichen Mitarbeiter erfolgreich durchzuführen.

List of Publications

International journals

1. Matteo Saveriano, Fabian Hirt, and Dongheui Lee, Human-aware motion reshaping using dynamical systems, *Pattern recognition letters*, 2017.

2. Dongheui Lee, Raffaele Soloperto, and Matteo Saveriano, Bidirectional Invariant Representation of Rigid Body Motions and its Application to Gesture Recognition and Reproduction, *Autonomous Robots*, 2017.

3. Pietro Falco, Matteo Saveriano, Eka Gibran Hasany, Nicholas H. Kirk, and Dongheui Lee, A Human Action Descriptor based on Motion Coordination, *Robotics and Automation Letters*, 2(2):8119-818, 2017.

Peer-reviewed conferences

1. Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee, Data-Efficient Control Policy Search using Residual Dynamics Learning, In *International Conference on Intelligent Robots and Systems*, 2017.

2. Riccardo Caccavale, Matteo Saveriano, Giuseppe A. Fontanelli, Fanny Ficuciello, Dongheui Lee, and Alberto Finzi, Imitation Learning and Attentional Supervision of Dual-Arm Structured Tasks, In *International Conference on Development and Learning and on Epigenetic Robotics*, 2017.

3. Caroline Blocher, Matteo Saveriano, and Dongheui Lee, Learning Stable Dynamical Systems using Contraction Theory, In *International Conference on Ubiquitous Robots and Ambient Intelligence*, pages 124–129, 2017. **Outstanding Paper Award.**

4. Roman Roor, Jonas Hess, Matteo Saveriano, Michael Karg, and Alexandra Kirsch, Sensor Fusion for Semantic Place Labeling, In *International Conference on Vehicle Technology and Intelligent Transport Systems*, pages 121–131, 2017. **Best Student Paper Award Candidate.**

5. Dharmil Shah, Pietro Falco, Matteo Saveriano, and Dongheui Lee, Encoding Human Actions with a Frequency Domain Approach, In *International Conference on Intelligent Robots and Systems*, pages 5304–5311, 2016.

6. Nicholas H. Kirk, Karinne Ramirez–Amaro, Emmanuel Dean–Leon, Matteo Saveriano, and Gordon Cheng, Online Prediction of Activities with Structure: Exploiting Contextual Associations and Sequences, In *International Conference on Humanoid Robots*, pages 744–749, 2015.

7. Matteo Saveriano, Sangik An, and Dongheui Lee, Incremental Kinesthetic Teaching of End-Effector and Null-Space Motion Primitives, In *International Conference on Robotics and Automation*, pages 3570–3575, 2015.

8. Raffaele Soloperto, Matteo Saveriano, and Dongheui Lee, A Bidirectional Invariant Representation of Motion for Gesture Recognition and Reproduction, In *International Conference on Robotics and Automation*, pages 6146–6152, 2015.

9. Matteo Saveriano and Dongheui Lee, Learning Motion and Impedance Behaviors from Human Demonstrations, In *International Conference on Ubiquitous Robots and Ambient Intelligence*, pages 368–373, 2014.

10. Matteo Saveriano and Dongheui Lee, Distance based Dynamical System Modulation for Reactive Avoidance of Moving Obstacles, In *International Conference on Robotics and Automation*, pages 5618–5623, 2014.

11. Vito Magnanimo, Matteo Saveriano, Silvia Rossi, and Dongheui Lee, A Bayesian Approach for Task Recognition and Future Human Activity Prediction, In *International Symposium on Robot and Human Interactive Communication*, pages 726–731, 2014. **Kazuo Tanie Award Candidate.**

12. Matteo Saveriano and Dongheui Lee, Invariant Representation for User Independent Motion Recognition, In *International Symposium on Robot and Human Interactive Communication*, pages 650–655, 2013.

13. Matteo Saveriano and Dongheui Lee, Point Cloud based Dynamical System Modulation for Reactive Avoidance of Convex and Concave Obstacles, In *International Conference on Intelligent Robots and Systems*, pages 5380–5387, 2013.

14. Francesco Donnarumma, Vincenzo Lippiello, and Matteo Saveriano, Fast Incremental Clustering and Representation of a 3D Point Cloud Sequence with Planar Regions, In *International Conference on Intelligent Robots and Systems*, pages 3475–3480, 2012.

International workshops

1. Riccardo Caccavale, Alberto Finzi, Dongheui Lee, and Matteo Saveriano, Integrated Task Learning and Kinesthetic Teaching for Human-Robot Cooperation, In *Italian Workshop on Artificial Intelligence and Robotics*, 2016.

2. Florian Winter, Matteo Saveriano, and Dongheui Lee, The Role of Coupling Terms in Variable Impedance Policies Learning, In *International Workshop on Human-Friendly Robotics*, 2016.

3. Riccardo Caccavale, Alberto Finzi, Dongheui Lee, Enrico Leone, Silvia Rossi, Matteo Saveriano, and Mariacarla Staffa, Integrating Multimodal Interaction and Kinesthetic Teaching for Flexible Human-Robot Collaboration, In *International Workshop on Human-Friendly Robotics*, 2015.

4. Jie Liu, Matteo Saveriano, and Dongheui Lee, Robot Compliant Behavior through Reinforcement Learning, In *International Workshop on Human-Friendly Robotics*, 2014.

5. Matteo Saveriano and Dongheui Lee, Invariant Representation of Motion for Gesture Recognition in Daily Life Scenarios, In *International Workshop on Human-Friendly Robotics*, 2013.

6. Matteo Saveriano and Dongheui Lee, Distance based Dynamical System Modulation for Reactive Collision Avoidance, In *DGR-Tage (DGR-days)*, 2013.

Posters and videos

1. Matteo Saveriano and Dongheui Lee, Safe Motion Generation and Online Reshaping using Dynamical Systems, In *International Conference on Ubiquitous Robots and Ambient Intelligence*, 2014. **Best Poster/Video Presentation Award.**

2. Sven Parusel, Hannes Widmoser, Saskia Golz, Tobias Ende, Nico Blodow, Matteo Saveriano, Alexis Maldonado, Ingo Kresse, Roman Weitschat, Dongheui Lee, Michael Beetz, Alin Albu-Schaeffer, and Sami Haddadin, Human-Robot Interaction Planning, In *AAAI Conference on Artificial Intelligence*, 2014. **Nomination for Best Robot Video Award.**

Nomenclature

Abbreviations

CLF	Control Lyapunov Function
CPU	Central Processing Unit
DMP	Dynamic Movement Primitives
DS	Dynamical System
DTW	Dynamic Time Warping
EM	Expectation Maximization
GAS	Globally Asymptotically Stable
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
GP	Gaussian Process
HMM	Hidden Markov Model
LAS	Locally Asymptotically Stable
LWPR	Locally Weighted Projection Regression
SEDS	Stable Estimator of Dynamical Systems

Conventions

$\boldsymbol{a} \cdot \boldsymbol{b}$	Scalar product
$\boldsymbol{a} \times \boldsymbol{b}$	Cross product
$\boldsymbol{A}$	Matrix
$\boldsymbol{a}$	Vector
$\boldsymbol{a}^{\mathrm{T}}$, $\boldsymbol{A}^{\mathrm{T}}$	Transpose of vector, matrix
$\boldsymbol{A}^{-1}$	Inverse of matrix
$\boldsymbol{A}^{\dagger}$	Moore–Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{f}(\cdot)$	Vector function
$\ddot{\boldsymbol{a}}$	Second-order time derivative
$\dot{\boldsymbol{a}}$	First-order time derivative
$\hat{\boldsymbol{a}}$	Unit vector
$\mathbb{R}$	Set of real numbers
$\mathcal{N}(\cdot)$	Gaussian function
$\lVert \cdot \rVert$	Norm
a, A	Scalar
$f(\cdot)$	Scalar function

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1. Motivations

The integration of robotic devices in human populated environments will greatly simplify everyone's life. Robots, in fact, will substitute humans in dangerous or tedious work, and will act as co-workers in heavy tasks. In order to pursue this objective, robots will be asked to autonomously accomplish daily-life activities and to actively cooperate with humans in dynamically changing environments. Such a tight human–robot cooperation requires the ability of the robot to continuously acquire novel skills. Moreover, the robot has to be able to adapt the task execution to changes in the environment, and to human intentions and needs.

A fruitful cooperation between human and robot workers requires several ingredients. Following a bottom up approach, the first condition for a proficient human–robot interaction is to guarantee the user safety. In a human–robot interaction scenario, human safety is ensured by equipping robotic devices with intrinsically safe, compliant actuators [57], or by exploiting collision detection and reaction mechanisms [58,60]. Apart from reducing the severity of possible injuries due to accidental collisions with the robot, preventing such collisions is also useful to increase the human confidence in the robotic partner. Moving one step up, the study in [88] shows that a human-aware adaptation of robot movements is beneficial for the human partner, since it improves the working speed and the human satisfaction. Hence, beyond the safety, a fruitful human–robot cooperation requires the adaptation of the robotic task to human intentions and needs.

A general purpose automaton, like a robot should be, is able to execute a variety of tasks and to continuously increase is knowledge and competences. As humans beings do, a general purpose robot is able to acquire novel skills by observing a demonstration of the task. This is the objective of the Programming by Demonstration (PbD) framework [14]. In PbD, in fact, an expert teacher shows how to perform a certain task, and the robot leverages this demonstration to acquire and then reproduce a novel skill. The procedure for skill transferring is natural and intuitive, since the user has only to demonstrate the task, for example by physically guiding the robot towards the task completion. However, general purpose machines are asked to execute "complex" tasks which involve the execution of several action and the manipulation of different objects. The actions constituting a complex task have to be identified and separated into the task demonstration, opportunely represented in the robot memory, and quickly adjusted to cope with eventual differences between demonstration and execution scenarios.

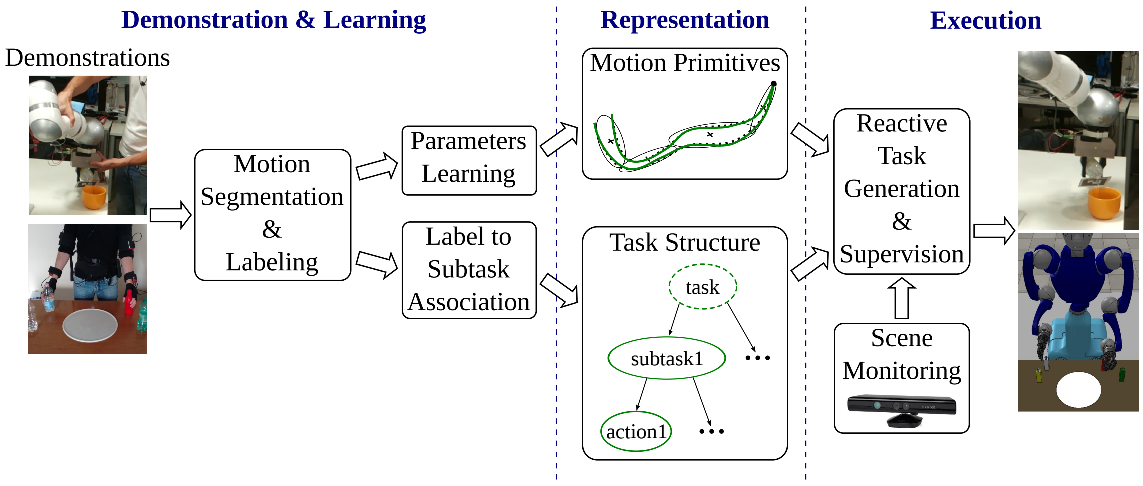


Figure 1.1.: Overview of this thesis work. (Demonstration & Learning) Human teachers intuitively transfer skills to robotic devices by using kinesthetic teaching or imitation learning. Demonstrated tasks are autonomously segmented and labeled. Segmented data are used to learn parameters of the selected motion representation via machine learning algorithms. Action labels are automatically attached to a partially specified task structure. (Representation) The selected motion representation and the learned parameters constitute a motion primitive which is used at run time to generate the robot's trajectory. The task structure resulting from the association of action labels and predefined subtasks is a symbolic representation of the task. (Execution) Task structure and motion primitives are used to reproduce learned tasks. Motion trajectories and task plan can be reactively adapted to cope with eventual changes in dynamic environments.

This thesis proposes a framework for intuitive transfer of complex tasks from a human teacher to a robotic learner. The framework allows a human teacher to demonstrate the execution of a task consisting of several actions and objects to manipulate. Moreover, the presented framework permits an autonomous execution of the acquired task in dynamic environments, as well as a fluent and smooth human–robot cooperation. To develop such a framework, one has to consider three aspects of the task acquisition process, namely:

- *Representation* - How complex tasks can be efficiently represented by the robot?
- *Learning* - How complex tasks can be intuitively and quickly transferred from a human to a robotic device?
- *Execution* - How complex tasks can be effectively executed autonomously or in cooperation with a human co-worker, considering both the human safety and the correct task execution as constraints?

Figure 1.1 shows an overview of the developed framework. Task demonstrations are firstly segmented into basic motion units and labeled. Segmented data are used to represent the demonstrated task in a way that allows the robot to retrieve and execute the task. Ideally, the representation has to be compact to reduce memory requirements, generalizable to execute the same task in different contests, and quickly accessible to rapidly generate the task plan. During the task execution, the robot's workspace is monitored to adapt the robot's trajectory to sudden changes in the scene, and to modify the task plan in case of human intervention.

1.2. Structure of the thesis

The rest of this thesis is composed of four chapters and two appendices. A brief overview of each chapter is outlined below.

Chapter 2

This chapter presents the related work and the contributions of this thesis. The chapter is organized in four main sections. The first section describes state-of-the-art approaches for robotic skills representation. The second section presents approaches for robotic skills learning from human demonstrations. The third section describes approaches for skills execution on real robotic devices. The major differences between the approaches presented in this work and prominent approaches in the literature are also underlined. Finally, the last section presents the contributions of this work.

Chapter 3

This chapter discusses approaches useful to represent basic robotic skills and structured tasks. The first section describes and compares approaches for point-to-point motion representation using stable dynamical systems. The second section focuses on transforming Cartesian trajectories into a space which exhibits some invariance properties, like the invariance to the field of view or the duration of the motion. Two invariant representations of rigid body motions are presented, namely a unidirectional representation and a bidirectional one. Several experiments are conducted to demonstrated the effectiveness of invariant trajectory descriptors in motion recognition and reproductions problems. The last section discusses the representation of structured tasks.

Chapter 4

This chapter presents approaches for intuitive transfer of tasks from a human instructor to a robotic device. The chapter focuses first on basic uni-manual skills and presents an effective solution to teach and to incrementally refine end-effector and null-space movements. The second section presents a kinematic approach useful to solve the motion re-targetting problem, that is the problem of transferring the motion between two agents with a different kinematic structure. The approach is effectively applied to imitate the human movements with a humanoid robot. The last section is dedicated to structured tasks, and presents a framework that combines learning from demonstration and attentional supervision to intuitively teach structured robotic tasks.

Chapter 5

This chapter focuses on executing a learned task on a real robot. The first section presents a hierarchical architecture that exploits stable dynamical systems and human monitoring to generate, and on-line replan, the robot's trajectory. The presented replanning strategies are able to generate collision-free trajectories, without affecting the execution of the robotic task. The distance between the robot and eventual obstacles in the scene is computed in real-time using an highly parallelizable algorithm, also detailed in this chapter. The hierarchical architecture is tested in a human–robot interaction scenario, showing promising results. The second part of this chapter discusses the execution of structured tasks. The framework exploited in Chapter 4 for task learning is here employed to generate the task plan for the robot and execute the task. The effectiveness of the framework is demonstrated in typical kitchen scenarios.

Chapter 6

This chapter states the conclusions of this work and summarizes the main finding. The chapter also provides final remarks on the presented approaches and proposes some research lines to further improve this work.

Appendix A

This appendix provides the theoretical background of several approaches adopted in this work. The appendix starts with a recap on dynamical systems and describes Lyapuniv and Contraction stability theories. Then, the appendix describes state-of-the-art algorithms in robot kinematic and dynamic control. These approaches are effectively exploited in order to run experiments on real robots. Finally, the appendix discusses three prominent approaches for non-linear regression, which are used in this work to compactly represent human demonstrations.

Appendix B

This appendix presents a theoretical and practical comparison between three invariant representations, namely the representation presented in Section 3.2.2, and those proposed in [155] and [42] respectively. Similarity and differences between the approaches are theoretically derived and shown with experiments on an artificial Cartesian trajectory.

1.3. Publication note

Most of the material presented in Chapters 3 to 5 and in Appendix B have been published in or submitted to peer-reviewed conference proceedings and scientific journals. References to the related publications for each chapter are given below.

Chapter 3

Caroline Blocher, Matteo Saveriano, and Dongheui Lee, Learning stable dynamical systems using contraction theory, In *International conference on ubiquitous robots and ambient intelligence*, 2017. **Outstanding Paper Award.**

Dongheui Lee, Raffaele Soloperto, and Matteo Saveriano, Bidirectional invariant representation of rigid body motions and its application to gesture recognition and reproduction, *Autonomous robots*, 2017.

Raffaele Soloperto, Matteo Saveriano, and Dongheui Lee, A bidirectional invariant representation of motion for gesture recognition and reproduction, In *International conference on robotics and automation*, pages 6146–6152, 2015.

Matteo Saveriano and Dongheui Lee, Invariant representation for user independent motion recognition, In *International symposium on robot and human interactive communication*, pages 650–655, 2013.

Matteo Saveriano and Dongheui Lee, Invariant Representation of Motion for Gesture Recognition in Daily Life Scenarios, *International Workshop on Human-Friendly Robotics*, 2013.

Riccardo Caccavale, Matteo Saveriano, Andrea Fontanelli, Fanny Ficuciello, Dongheui Lee, and Alberto Finzi, Imitation learning and attentional supervision of dual-arm structured tasks, In *International conference on development and learning and on epigenetic robotics*, 2017.

Riccardo Caccavale, Alberto Finzi, Dongheui Lee, and Matteo Saveriano, Integrated Task Learning and Kinesthetic Teaching for Human-Robot Cooperation, *Italian Workshop on Artificial Intelligence and Robotics*, 2016.

Chapter 4

Matteo Saveriano, Sangik An, and Dongheui Lee, Incremental Kinesthetic Teaching of End-Effector and Null-Space Motion Primitives, In *International Conference on Robotics and Automation*, pages 3570–3575, 2015.

Riccardo Caccavale, Matteo Saveriano, Andrea Fontanelli, Fanny Ficuciello, Dongheui Lee, and Alberto Finzi, Imitation learning and attentional supervision of dual-arm structured tasks, In *International conference on development and learning and on epigenetic robotics*, 2017.

Riccardo Caccavale, Alberto Finzi, Dongheui Lee, and Matteo Saveriano, Integrated Task Learning and Kinesthetic Teaching for Human-Robot Cooperation, *Italian Workshop on Artificial Intelligence and Robotics*, 2016.

Chapter 5

Matteo Saveriano, Fabian Hirt, and Dongheui Lee, Human-aware motion reshaping using dynamical systems, *Pattern recognition letters*, 2017.

Matteo Saveriano and Dongheui Lee, Distance based dynamical system modulation for reactive avoidance of moving obstacles, In *International conference on robotics and automation*, pages 5618–5623, 2014.

Matteo Saveriano and Dongheui Lee, Safe Motion Generation and Online Reshaping using Dynamical Systems, *International Conference on Ubiquitous Robots and Ambient Intelligence*, 2014. **Best Poster/Video Presentation Award.**

Matteo Saveriano and Dongheui Lee, Point cloud based dynamical system modulation for reactive avoidance of Convex and concave obstacles, In *International conference on intelligent robots and systems*, pages 5380–5387, 2013.

Riccardo Caccavale, Matteo Saveriano, Andrea Fontanelli, Fanny Ficuciello, Dongheui Lee, and Alberto Finzi, Imitation learning and attentional supervision of dual-arm structured tasks, In *International conference on development and learning and on epigenetic robotics*, 2017.

Riccardo Caccavale, Alberto Finzi, Dongheui Lee, and Matteo Saveriano, Integrated Task Learning and Kinesthetic Teaching for Human-Robot Cooperation, *Italian Workshop on Artificial Intelligence and Robotics*, 2016.

CHAPTER 2

Background research and contributions

This chapter presents the literature related to this work and the contributions of this thesis. Related work are split into three main sections. The first section presents work related to the representation of robotic skills and tasks. The second introduces state-of-the-art approaches for skills and tasks learning from human demonstrations. The third section presents work on robotic tasks planning and execution. Apart from discussing related work, each section also underlines similarities and differences between prominent approaches in the filed and the solutions presented in this thesis. Finally, the last section summarizes the innovative contributions of this work.

2.1. Robotic tasks representation

2.1.1. Dynamical system based representations

Rough Cartesian or joint trajectories, i.e. sequences of points in joint or Cartesian space, are not well-suited for representing robotic skills. Rough trajectories, in fact, are hardly generalizable to different scenarios and require to store many data points. Hence, researchers in the field have focused on developing parameterized representations of robotic skills. In this way, a learned skill is represented in a compact form which allows the generalization to different scenarios and reduces memory requirements. Typical skill parameterizations include Gaussian mixture models (GMM/GMR) [18, 33, 75, 76], hidden Markov models [90, 91, 133], and Gaussian processes [78, 82]. In general, any regression technique [16] can be used to compactly represent robotic skills.

Regression algorithms can be used, for example, to describe the relationship between the current position in joint or Cartesian space and the relative velocity, i.e. the robotic skill is encoded into a dynamical system (see Appendix A.1). Stable dynamical systems (DS) are well-suited to represent the so-called point-to-point motions, which are movements starting from any initial point and terminating at a given target. Stable DS, in fact, are guaranteed to converge to a specified target. Moreover, dynamical systems are robust to changes in the initial/target location, and can be used in cluttered environments to generate collision-free paths, as discussed in Section 5.1. The dynamic movement primitives (DMPs) framework proposed by Ijspeert et al. [65–67] is one of the first examples of robotic skills representation via stable dynamical systems. A DMP is a superposition of a linear spring–damper dynamics and a non-linear forcing term learned from a single demonstration. A clock signal is used to suppress the non-linear force after a certain time guaranteeing the convergence towards the target. Task-parameterized motion primitives (TP-DMPs) [31] extend the standard DMPs by introducing extra task-dependent parameters useful to adapt robot movements to novel scenarios. TP-DMPs consist of a weighted summation of second order DS, learned from demonstrations using Gaussian mixture models (GMM) [39]. Probabilistic Movement Primitives (ProMPs) [113] represent another extension of the standard DMPs. The idea of ProMPs is to learn a distribution over multiple trajectories, which allows a better adaptation to novel, unforeseen situations. Several approaches extend the DMPs framework to incremental learning scenarios [55, 86, 105, 118]. In [55] authors present a two-layers system for incremental learning of periodic movements. The first layer of the system is a DS which extracts the fundamental frequency of the demonstrations. The second layer is a periodic DMP which learns the waveform of the demonstrated motion. The overall system is computationally efficient and works on-line, but it is limited to periodic motions, while discrete movements are the focus of this work. The work in [118] considers incremental human coaching for DMPs. In the teaching phase, the user is considered as an obstacle, avoided by adding an extra forcing term to the DMPs [62]. In this way, the human is able to modify on-line the robot's path without touching it. The novel path is used to incrementally update DMPs weights via recursive least square. [105] leverages iterative learning control [22] to realize a learning strategy which is faster and more robust than recursive least square. Both the approaches in [105, 118] are evaluated on periodic movements, but they are also applicable to point-to-point motions. The interaction between two agents, each with is own path defined by DMPs, is incrementally learned in [86] to guarantee that both agents equilibrate into a common target, i.e. the two agents are effectively helping each other.

DMPs are time dependent dynamical systems, since a clock signal is used to suppress the forcing term. This means that, from a certain time on, the robot will follow a linear dynamics towards the target. In some situations, it is useful to represent the motion via an autonomous (time independent) and non-linear system. Khansari–Zadeh and Billard propose effective solutions to generate stable motions from a non-linear DS represented by GMR [75, 76]. The binary merging approach in [75] uses GMM to determine if is the current state (robot position) is inside or outside the demonstration area. The asymptotic convergence to a given target is then guaranteed for trajectories inside the demonstration area. In other words, the approach in [75] guarantees local convergence to the target. This limitation is overcome by the stable estimator of dynamical systems (SEDS) [76], which guarantees global convergence of a system represented by GMR. In SEDS, Lyapunov theory (see Appendix A.1) is used to derive sufficient stability conditions for a system represented by GMR. The derived stability constraints are then used to learn the GMM parameters by solving a constrained optimization problem. The main advantage of SEDS is that the learned DS is proven to be globally stable. The drawback is that the stability constraints are derived from a quadratic Lyapunov function. For complex motions, contradictions may occur between demonstrations and quadratic stability constraints, which prevents SEDS to accurately learn the motion [78]. The accuracy problem is explicitly considered in several works [78, 93, 106, 117, 128], showing that complex motions can be accurately represented by non-linear DS. The work in [78, 93] propose two different approaches to learn a Lyapunov function which minimizes the contradictions between the stability constraints and the training data, guaranteeing an accurate reproduction of complex motions. [106] applies a diffeomorphic transformation, learned from demonstrations, to project the training data into a space where they are well represented by a quadratic Lyapunov function. The work in [117] propose a fast algorithm to learn diffeomorphic transformations from demonstrations. The approach is significantly faster than [106], but it works only for linear DS. The work in [128] exploits Contraction theory (see Appendix A.1) to stabilize a DS represented via a neural network. Authors derive sufficient stability

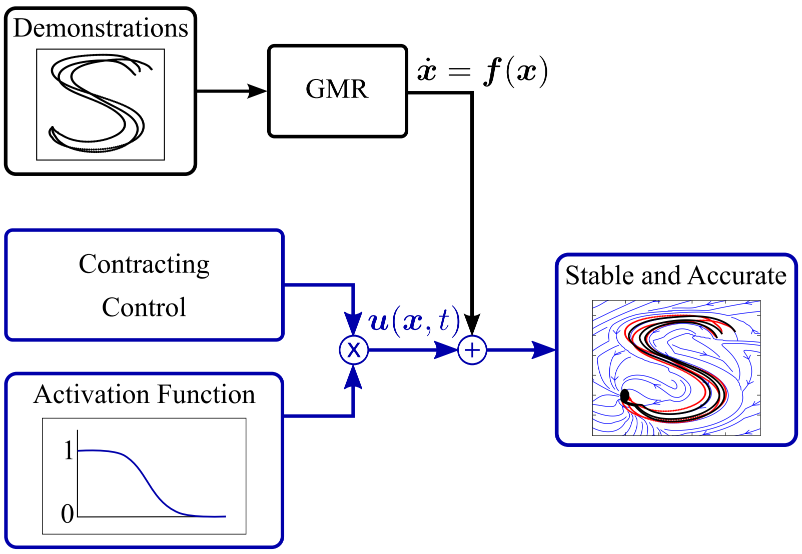

Demonstrations
GMR
$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$
Contracting Control
$\boldsymbol{u}(\boldsymbol{x}, t)$
Activation Function
1
0
Stable and Accurate

Figure 2.1.: Overview of the contracting Gaussian mixture regression approach presented in [18]. The original system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$, generated by Gaussian mixture regression, is not guaranteed to converge towards a unique equilibrium. This system is stabilized at run time by the control input $\boldsymbol{u}(\boldsymbol{x},t)$, automatically computed given the Gaussian mixture parameters. The control input is smoothly activated/deactivated by an activation function in order to guarantee accurate motion reproduction and convergence to the target.

conditions using the contraction analysis. Derived stability constraints are then used to learn the neural network parameters, by solving a constrained optimization problem.

As shown in [78], the numerical solution of a constrained optimization problem is computationally expensive. In order to reduce the computational cost, the approach in [78] exploits a two-steps learning procedure. Motion demonstrations, in fact, are used to learn both a (possibly) unstable DS and a control Lyapunov function [146]. The unstable DS is then stabilized at run time by applying a control input automatically computed from the learned control Lyapunov function. Notice that a two-steps learning procedure is also exploited in [106], where training data are used to learn both the diffeomorphism and the stable DS. Instead of applying a two-steps learning procedure, the *Contracting GMR* (C-GMR) approach in [18] exploits Contraction theory to derive sufficient stability conditions for a DS represented by GMR, and it leverages derived conditions to automatically compute a stabilizing control input. The control input only depends on the given GMR parameters. Hence, C-GMR requires only one learning step to fit the GMM parameters given the demonstrations. As shown in Figure 2.1, the control action is smoothly activated or deactivated using an activation function. This solution permits to improve the reproduction accuracy without affecting the stability of the controlled DS. Compared to [128], C-GMR performs the contraction analysis of a GMR system, and it uses the resulting constraints to compute, at run time, a stabilizing control input. C-GMR shares with SEDSII the idea of stabilizing the DS at run time, but it does not require the learning of a suitable Lyapunov function. The C-GMR approach is presented in Section 3.1.3, together with an extensive experimental evaluation and a comparison with Lyapunov based approaches in [76, 78].

2.1.2. Invariant representations

The previous section discussed the limitations of Cartesian trajectories in terms of high memory requirements and low generalization capabilities, and it suggested to overcome these limitations using dynamical system based representations. This section focuses on possible motion variations that may affect Cartesian trajectories, limiting the applicability of Cartesian descriptors in gesture recognition and reproduction problems. As underlined in [155], Cartesian trajectories are affected by two kinds of motion variations, depending on *how* and *where* the motion is executed. In particular, the motion variations depending on *how* the motion is executed are:

- *Execution time and speed* - The same action is slower or faster.
- *Amplitude* - The same action is longer or shorter.

The variations depending on *where* the motion takes place are:

- *Starting pose* - The same action is executed from different starting points.
- *Reference frame* - The sensor used to track the human, e.g., a camera, is moved across different repetitions.
- *Reference point* - A Cartesian trajectory is the motion of a reference point attached to the human (robot) body in a reference frame. The reference point can vary across different capturing sessions, for instance because a marker was attached in a slightly different position.

Presented motion variations complicate the problem of gesture recognition in real situations. A possible solution to increase the recognition performance consists in using an invariant form of motion trajectories, i.e. a coordinate-free and scale invariant representation of motion [43, 147]. Invariant trajectory descriptors free the Cartesian trajectory from motion variations, helping to focus on essential aspects of the motion. This is beneficial also for a flexible generation of robot motions, since the invariance to the starting pose and the amplitude of motion can be exploited to generate affine transformed instances of a motion trajectory, without providing further demonstrations. To this end, it is of importance to reconstruct a Cartesian trajectory form an invariant one, or, in other words, to have a bidirectional invariant representation.

As shown in Figure 2.2, unidirectional representations create a feature space with invariant properties, such as roto–translations and scale invariance, which is beneficial to increase the recognition rate in gesture recognition problems. The simplest way to obtain invariant features is to use the joint angles instead of the Cartesian positions. Joint angles, in fact, are naturally invariant to roto-translations and limb lengths. Full-body motion descriptors are built in [49, 111, 141] by using the joint angles among the human limbs, and exploited to recognize full-body motions. However, it is known that recognizing motions using only joint angle trajectories is a challenging problem [35]. Moreover, human tracking systems as RGB-D cameras directly provides the Cartesian pose of each limb, and the human kinematics is needed to extract the joint angles by using inverse kinematics algorithms (see Appendix A.2.2).

In order to overcome the limitations of angle based representations, several researchers focused on extracting invariant features from Cartesian trajectories. In [17] human motions are modeled as temporal trajectories of estimated parameters (state) over time. The state is used to control stretching, scaling, and translations of the gesture model with respect to the incoming data. The condensation algorithm [70] is then employed to incrementally match trajectory models to the multi-variate input data. The condensation algorithm does not make any parametric assumption about the state probability distribution and requires a large set of densities to estimate current parameters. To reduce the computational cost of the prediction, in [121] hidden Markov models

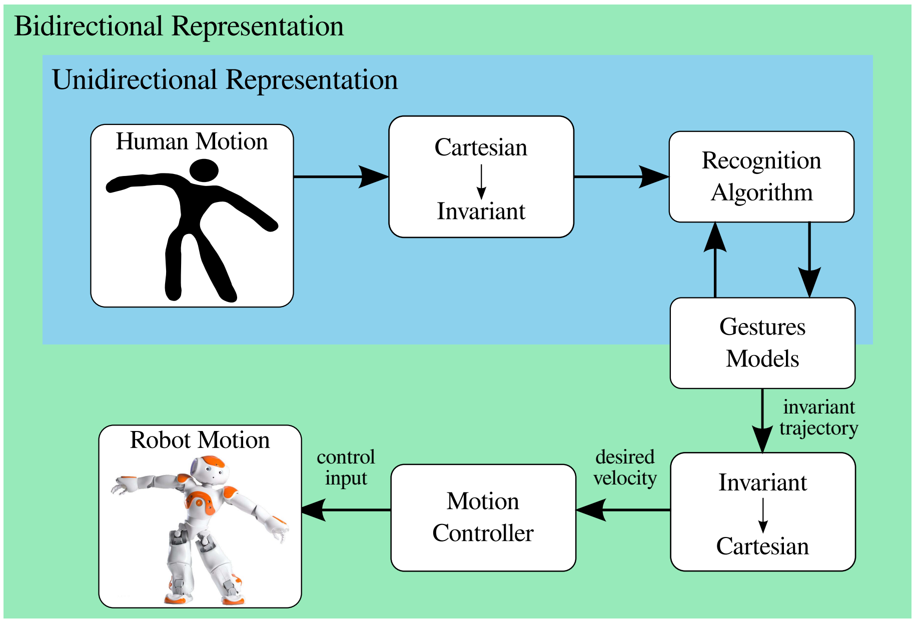


Figure 2.2.: Possible applications of uni- and bidirectional invariant representations. A unidirectional representation transforms motion trajectories into invariant descriptors, which serve as features for a motion recognition algorithm. A bidirectional representation allows the mapping form Cartesian to invariant space and vice versa. Hence, bidirectional representations can be also used to generate different Cartesian trajectories from the same invariant descriptor.

(HMM) [123] are used to learn the prior distribution of both the observation covariance and the state transition probabilities. Action descriptors invariant to affine transformations are directly computed from image coordinates in [119, 120, 160, 168]. The representation in [119, 120] is invariant under affine and projective transformations. Invariant values are computed by considering five fixed points on the rigid body and by tracking them in the image during the entire motion. Three dimensional invariants under affine transformations are proposed in [160, 168]. Their computation requires to track the same six points in all frames.

Alternative approaches focus on Euclidean group invariants. In [125, 126] gestures are represented by the spatio–temporal curvature of the Cartesian trajectory, that is invariant to roto–translations. This representation is useful to capture the dynamic instants of a trajectory, i.e. points where the velocity and/or the direction of motion change sign. The invariant descriptor in [135] consists of two values, which describe respectively the linear and angular part of the motion, and is invariant to roto–translations and linear scaling. For this descriptor, the scale invariance is not obtained by introducing an artificial scale depending on the motion length, but it holds for each frame (see Section 3.2.1). This approach is used in [97] to recognize human tasks by fusing gesture and object recognition into a dynamic Bayesian network [16]. The Fourier transform is leveraged in [157] to compute a view and scale invariant representation. Cartesian trajectories are partitioned into a temporal pyramid and the Fourier transform is computed for each segment at each level. Low-frequency Fourier coefficients are then used as feature vectors.

The 4D Action Feature Model (4D-AFM) is proposed in [166] to recognize full-body gestures observed from arbitrary views. Given a multi-view video sequence, a 3D shape for each frame is computed by projecting the multi-view 2D silhouettes into the 3D space. For each silhouette, a set of features (action sketch) describing changes in direction, speed, and shape of the contour is computed. The 4D-AFM consists of a sequence of 3D shapes with action sketches attached. Even if it is possible to recognize actions from a single view, multi-view video sequences are required to construct the model in the training process. In [94] actions are modeled as a graph, where each node represents a salient posture. Salient postures are described by a set of 3D points belonging to the human body. In [165] a histogram of 3D joint locations (HOJ3D) is created by dividing the space into bins. The HOJ3D is then projected into a lower dimensional space using linear discriminant analysis [16] and clustered into k classes. HMMs are used for gesture recognition.

The availability of large computational power and big data has recently made deep neural networks popular in different research fields. Local invariance to translations is guaranteed in convolutional neural networks (CNNs) by the convolutional and pooling layers [45]. CNNs are effectively used in [158] to recognize human actions from input depth maps. In [45] the standard CNN is extended with four extra layers (slice, pool, roll, and stack) which guarantee local invariance to rotations. Inspired by the recognition process in the visual cortex, [89] presents a hierarchical architecture to learn, from visual inputs, features invariant to affine transformations and illumination. Invariant features are learned in an unsupervised manner using sparse auto-encoders. [71] proposes to learn warping transformations from input data. This extra learnable layer can be inserted into CNNs to improve the recognition performance. Approaches based on deep learning have shown high recognition accuracy in typical computer vision problems, like object recognition and scene parsing, and have good potential in action recognition problems. Nevertheless, their applicability in motion generation problems is still limited. Indeed, even if neural networks can learn a set of (locally) invariant features, it is hard to reconstruct a Cartesian motion from these features. Moreover, deep learning approaches require long training time, big amount of data, and high computational power. This limits their applicability on autonomous robotic devices, which usually have limited computational power.

Presented unidirectional invariant representations are not suitable for motion reproduction, since the invariant trajectory cannot be transformed into a Cartesian one. As already mentioned, bidirectional invariant representations are effective motion descriptors which increase the recognition rate and allow the generation of different trajectory instances from the same descriptor. Frenet–Serret (FS) invariants [83] are the first example of a bidirectional invariant representation. FS representation uses curvature, torsion, and their first-order derivatives to describe the motion of a spatial curve. The approach in [163] computes an approximate version of the FS invariants by using points in consecutive time instants instead of high-order time derivatives. The resulting approximate FS representation is numerically robust, invariant to affine transformations and changes in the speed of execution. Original positions are retrieved by numerically integrating the Frenet–Serret equations [164]. FS invariants and their robust version in [163] neglect the orientation part of the motion, which is of importance for recognition and reproduction of complex motions. This limitation is overcome in [155], where Extended Frenet–Serret (EFS) invariants are proposed to consider also the orientation part of the motion. The EFS is not affected by affine transformations, time, linear, and angular scale, and motion profile. EFS invariants are compared with the bidirectional representation in [42], which is constructed by means of the Instantaneous Screw Axis (ISA) [101]. The comparison shows a superior recognition rate of the EFS invariants, due to the higher noise sensitivity of [42]. EFS invariants and the representation in [42] depend on high-order time derivatives of the velocity. The numerical computation of high-order derivatives on real, non-smooth signals is sensitive to noise and round-off errors [34]. As shown in Section 3.2.4 and Appendix B, the sensitivity of numerical derivatives makes difficult to estimate reliably the invariant values in [42, 155] and generates a non-negligible reconstruction error. These limitations are overcome by the Denavit–Hartenberg inspired Bidirectional (DHB) representation proposed in [92, 147]. DHB is a minimal (six values) and numerically robust invariant repre-

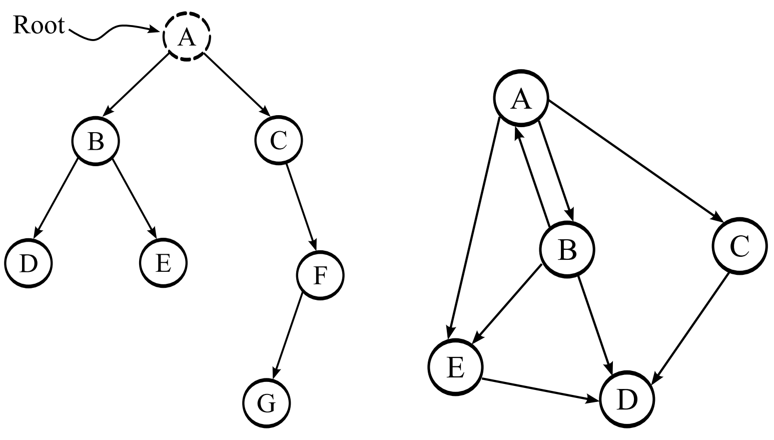


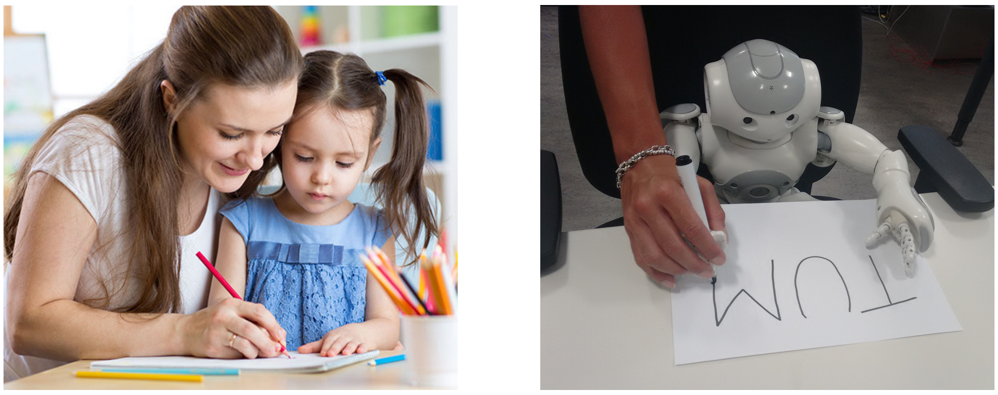
Figure 2.3.: An example of tree and graph based representations. (a) The tree is a hierarchical model with a single root node and no loops between the nodes. (b) The graph is a network model where all the nodes are at the same level and loops between nodes are allowed.

sentation computed by the means of the Denavit–Hartenberg notation. Similarly to [163] and in contrast to [42, 155], DHB exploits points in consecutive time instants instead of high-order time derivatives. In contrast to [163], DHB also considers the orientation part of the motion. The DHB representation is theoretically presented in Section 3.2.2. Experiments and comparisons in motion recognition and reproductions problems are presented in Section 3.2.3 and 3.2.4 respectively. Finally, Appendix B theoretically compares DHB and the invariants in [42,155] in order to underline similarities and differenced among the three representations.

2.1.3. Representation of structured tasks

Representations presented in the previous sections focus on describing a robotic skill consisting of a single movement, like a point-to-point motion. In general, robots are asked to solve more complex tasks involving a sequence of actions to perform on some objects in the scene. It is clear that such "complex" tasks require a customized representation that allows task storage and generation. In particular, the task structure must contain all the information needed to generate the learned task. The information include actions, objects, eventual properties, and execution constraints. Several approaches in the robotics literature adopt topological representations of complex tasks, such as oriented graphs [46, 72, 99, 107, 108, 124] and rooted trees [19, 26–28, 30]. An example of three and graph structures is shown in Figure 2.3.

As detailed in Section 3.3, this work exploits a rooted tree to represent a structured task. The choice is a direct consequence of the definition of structured task. Indeed, in this work, the term structured task is used to indicate a task which involves the execution of multiple actions on different objects, like preparing a certain receipt. Moreover, a structured task can be hierarchically decomposed into different subtasks. For example, the task of putting a tee bag in a cup with hot water can be decomposed into two subtasks, namely take the tea bag from the box and put it into the cup. Each subtask can be also decomposed into a series of basic actions. For instance, the subtask of taking the tea bag contains the elementary actions reach the tea box, open the box (if closed), and grasp the tea bag. These actions are effectively commanded to the robot. Hence, no further decompositions of the elementary actions is possible. Considering this definition of structured task, it is clear that the topological representation of a structured task requires a root node



(a) Human–human kinesthetic teaching. (b) Human–robot kinesthetic teaching.

Figure 2.4.: Kinesthetic teaching, i.e. physically guiding the learning subject towards the task execution, is a natural and intuitive way that humans use to quickly transfer skills among each other. (a) Image downloaded from http://www.momjunction.com/articles/fun-ways-to-teach-your-toddler-to-write-better_0082821/#gref in September 2017.

(put tea in a cup), one or multiple intermediate layers (take the tea bag), and several leaves (reach the tea box, open the tea box). Hence, a rooted tree is well-suited to represent a structured task.

2.2. Robotic tasks learning

Robots are required to rapidly acquire new skills and functionalities. Hand programming of robotic tasks is no longer an option to increase the robot capabilities, since it requires significant programming effort and expert knowledge. The Programming by Demonstration (PbD) framework [14] represents a valid alternative to rapidly increase robot skills, without the need of tedious and time consuming hand programming. In the PbD framework, in fact, the user teaches the robot how to perform a certain task by demonstrating the correct task execution (see Figure 2.4). Collected demonstrations are then mapped into robot movements and eventually represented in a compact form, as discussed in the previous section. This section focuses on two aspects of the PbD paradigm which are further investigated in Chapter 4, namely how to incrementally acquire/refine novel skills and how to learn structured tasks from human demonstrations.

2.2.1. Incremental skills learning from demonstrations

Two main ingredients are required to incrementally refine novel skills. First, the algorithm used to represent the demonstrated skill has to permit an incremental refinement of the learned parameters (see Section 2.1). Second, the user has to provide novel task demonstrations, possibly during the task execution. Indeed, providing novel demonstrations during the execution is beneficial to teach natural and coordinated movements, and it gives the user the possibility to refine only part of the motion, leaving the rest unchanged. It is clear that the gravity compensation control (see Appendix A.2.3), widely used to collect demonstrations via kinesthetic teaching, is not sufficient to provide kinesthetic demonstrations at run time. This problem is solved in [91] by exploiting a customized variable impedance control. In particular, the robot has high stiffness around the nominal trajectory in order to accurately track the desired path in free motion. Eventual forces generated by the interaction with the user deviate the robot from the nominal path and the stiffness is decreased accordingly to favor an easy guidance. The stiffness is increased again if the robot deviates too much from the nominal trajectory. In this way, the user feels a soft constraint which prevents too large deviations from the nominal path.

In case of redundant manipulators, which are robots with more degrees-of-freedom (DoF) than the ones required for task execution, the problem arises of exploiting redundant DoF in a fruitful manner. As known from the robot control literature, in fact, redundant DoF can be used to execute multiple tasks at the same time. Null-space projection techniques are usually adopted to avoid that lower priority tasks affect the execution of the higher priority ones [10, 36, 103]. From the learning perspective, null-space and end-effector tasks can be learned from human demonstrations. In [109], a recurrent neural network is used to learn the inverse kinematic mapping and null-space constraints from recorded end-effector positions and joint angles. Training data are collected in two steps. Firstly, the user moves the robot to the desired end-effector position. During the motion, the robot's configuration is adapted to consider eventual constraints in the workspace. Then, with the end-effector fixed in the desired position, the user teaches some local null-space configurations. End-effector and null-space tasks are then executed using a multi-priority controller. In [151] null-space policies are learned from observations considering the first priority task as a set of constraints on the null-space policy. The null-space policy is then estimated by solving an optimization problem, where the cost function to minimize represents the inconsistency among the observations. These approaches are effective in learning end-effector and null-space policies, but they do not allow the on-line refinement of learned null-space movements.

A unified framework to incrementally learn end-effector and null-space motions is proposed in [133] and presented in Section 4.1. The approach exploits a multi-priority kinematic controller to allow motion execution and kinesthetic teaching. In particular, the controller considers trajectory tracking and physical guidance as prioritized tasks, and it is capable of smoothly and continuously changing the priority between them. This allows the kinesthetic teaching of null-space tasks without affecting the end-effector motion, as well as the incremental refinement of the end-effector movement. End-effector and null-space skills are encoded into hidden Markov models, and the approach presented in [91] and summarized in Appendix A.3.3 is used to incrementally refine the HMM parameters as novel demonstrations are provided.

2.2.2. Learning structured tasks from demonstrations

Kinesthetic teaching and imitation learning are effective to transfer elementary motions or motion primitives from a human demonstrator to a robotic device. In order to apply these intuitive transfer techniques to structured tasks, it is needed to apply a segmentation strategy that separates the task demonstrations into elementary motions. Fod et al. [53] define a segmentation strategy *effective* if: *i)* it is fast enough to work in real-time, *ii)* it is consistent across different demonstrations of the same task, and *iii)* it is complete, meaning that the generated segments represent the entire task. The popular segmentation strategy in [53] suggests to segment an action stream looking at the zeros in the joint velocities. Velocities smaller than a given threshold value are considered as zero. In [85, 149] effective strategies are proposed for human motion segmentation into atomic motion units. The data stream is split into smaller units of fixed length [149] or using a moving window of fixed size [85]. Hidden Markov models are then used to recognize and reproduce the motion units. The performance of these approaches depends on some tunable parameters, i.e. the units length in [149], the window size in [85], and the threshold value in [53]. Depending on the type of motion, a certain value of the parameters can generate an oversegmentation, where many segments contains only few frames. Moreover, the approaches in [53, 85, 149] are effective in segmenting free human motions, but they do not provide a matching between action segments and objects in the scene. In order to properly learn structured tasks, each action has to be associated

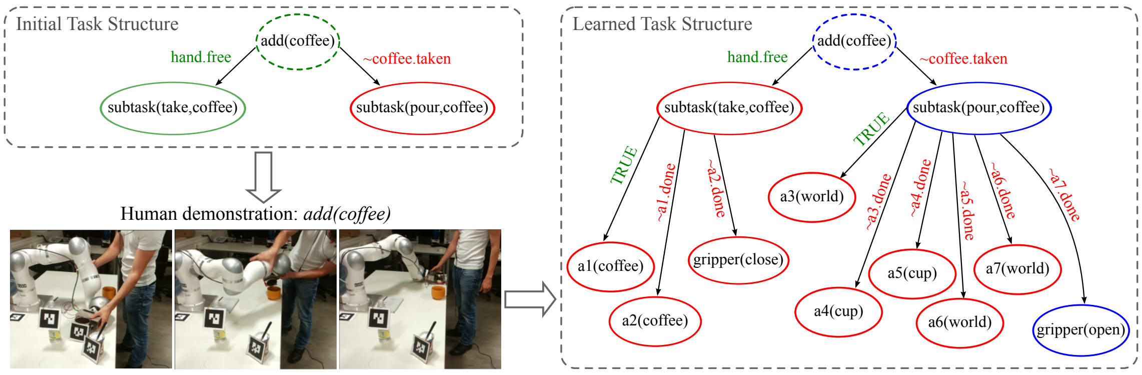


Figure 2.5.: The robot learns how to *add(coffee)* in a cup. In the tree structures, a red ellipse indicates an inactive node, a green ellipse indicates a node that is ready to be activated, and a blue ellipse indicates an active node. (Top left) The initial task structure consists of an abstract task *add(coffee)* and two subtasks. The *subtask(take,coffee)* is green because it has a true pre-condition (*hand.free*). (Bottom left) Snapshots of the task demonstration. (Right) The task structure learned from the human demonstration using the approach presented in Section 4.3.3. Each action a_i is a label (symbol) associated to a learned motion primitive. The action *gripper(open)* is blue because it is the last activated (learned) action.

with an object in the scene. The relation between objects in the scene and actions to perform is considered in [156], where object distances are exploited to split the demonstrated task into action segments. The approach can effectively generate basic actions with associated objects, but it requires a library of predefined object–action complexes [161] to reproduce the segmented task with a real robot. Following the approach in [156], this work exploits a simple and effective segmentation mechanism based on object proximity and explicit human commands. Each object in the environment is associated with a proximity area, which is a sphere of radius r around each object. Novel segments are generated when the end-effector(s) of the robot enters or leaves the proximity area of an object, or when an explicit human command is executed. The approach is exploited in [30] to segment dual-arm tasks, and it is described in Section 4.3.1.

The output of the segmentation strategy are a set of motion trajectories, each associated to an object in the scene. A unique label is also associated to each segmented movement. As discussed in Section 2.1, each motion trajectory is represented in a compact form used to retrieve the robot's trajectory at run time. This low level representation is not sufficient to fully describe a structured task, where the actions have to be executed with a certain order or on a specific object. Hence, an abstract description of the task, the so-called *task structure*, is required to properly learn and execute the task. As already mentioned, the natural way to represent a structured task is a tree based structure with logical pre- and post-conditions, labels, weights, and relative objects associated to each node. As shown in Figure 2.5, the task structure is (partially) learned from demonstrations using the approach described in Section 4.3.3. In particular, given an initial task structure (see the top left panel in Figure 2.5), an attentional supervisory system [26, 27, 110] is exploited to attach the generated segment (action) to the most emphasized subtask. Each action is associated to pre- and post-conditions which regulate the task execution.

The approach presented in this work focuses on autonomous learning of structured tasks from human demonstrations. As already mentioned, this involves the automatic segmentation of task demonstrations, the representation of segmented data in a form useful to generate motor commands for the robot, and the learning of a high-level structure required to generate the task plan.

Similarly, the work in [99, 108, 115] focus on learning a set of motion primitives from multiple demonstrations while automatically organizing them into graphs or automata. These approaches permit to learn and reproduce complex robotic tasks from human demonstrations. However, differently from the approach in this work, they do not consider the possibility of executing the learned tasks in cooperation with the human.

Alternative work focus on the problem of learning high-level task representations from human observations. In [150, 169] sequential constraints (like reaching an object and then grasping it) are used to find a set of semantic rules that determine the sequence of actions to perform. Semantic rules are also used in [124] to learn, recognize, and reproduce human activities from video sequences. Human activities are segmented using the zero velocity crossing approach in [53]. The segmented activities are then matched with a set of pre-programmed motion primitives and executed by the robot. The problem of task learning from the observation of human activities is also faced in [46]. Here, the human demonstration is used to generate a robot-independent task structure associated with robot-specific primitives. Similarly, in [72] a graph structure that represents bi-manual tasks is learned from human observations. Aforementioned approaches are effective in learning the task structure from human observations, but motion primitives are assumed as given. The approach presented in this work is complementary, it assumes that an abstract description of the task is available, while the goal is to learn both the motion primitives and their relations with the task structure.

In [107] the teacher uses simple verbal cues to facilitate the learning process. In particular, the authors propose explicit verbal instructions to bias the learner's attention to relevant aspects of the demonstration. Differently from this approach, the framework exploited in this work adopts a supervisory attentional system that enables more complex attention-base interaction (verbal and non-verbal) during both the teaching and execution phases. Social attentional mechanisms for non-verbal task teaching are investigated in [21]. In this case, the authors mainly focus on visual attention and gaze direction. In particular, they show the effectiveness of spatial scaffolding cues during interactive task demonstrations. Visual attention mechanisms for robot learning are also proposed in [13, 19, 102]. In contrast to these work, this thesis focuses on executive attention and cognitive control mechanisms supporting kinesthetic task teaching. Supervisory attentional frameworks for robotic systems have been proposed in [74] considering also cooperative tasks execution [25, 27], but not in a learning by demonstration context. Attentional mechanisms have been employed for robot teaching [19, 21] and imitation learning [148]. However, the framework presented in this work fully integrates attentional mechanisms within a supervisory attentional system paradigm [40, 110].

2.3. Robotic tasks execution

Presented approaches for task representation and learning constitutes the preliminary steps that permit the task execution on real robots. Given a learned structured task, the problems arise of selecting the next action to execute (task planning) and of executing the commanded action in partially unknown and potentially dynamic environments (see Figure 2.6). The rest of the section describes state-of-the-art work in these fields and underlines similarities and differences with the approaches presented in Chapter 5.

2.3.1. Reactive motion planning

Robots have to execute learned movements in dynamically changing environments and in close interaction with human operators. This requires a quick adaptation of the robot behavior to sud-

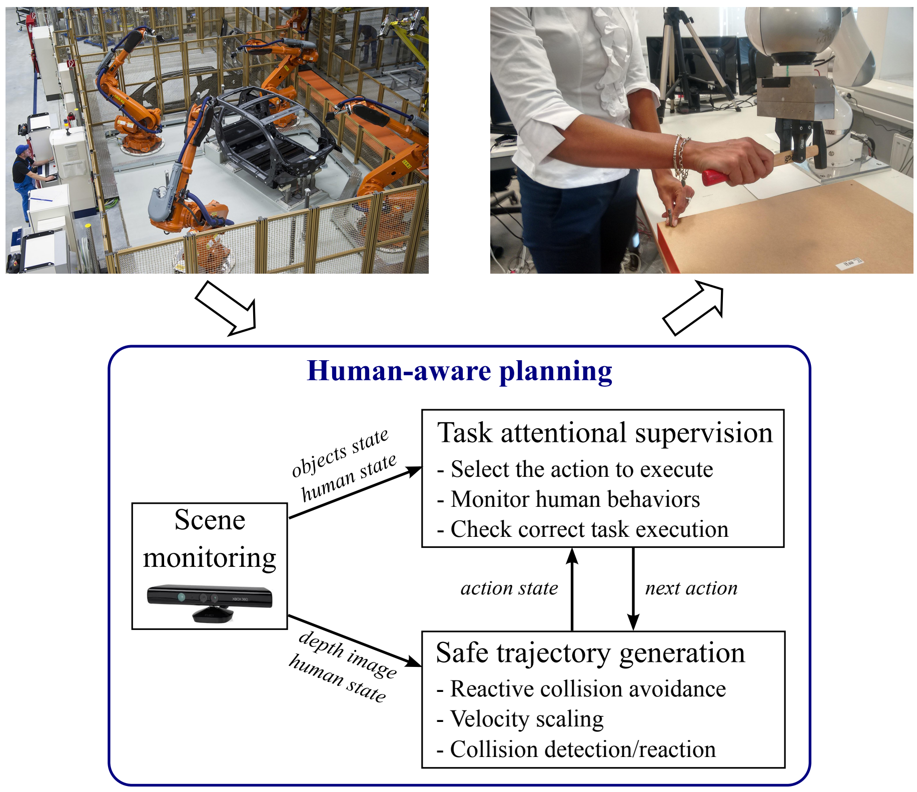

Human-aware planning
Scene monitoring
objects state
human state
Task attentional supervision
- Select the action to execute
- Monitor human behaviors
- Check correct task execution
action state
next action
depth image
human state
Safe trajectory generation
- Reactive collision avoidance
- Velocity scaling
- Collision detection/reaction

Figure 2.6.: A barrier-free human–robot cooperation has to take into account both human safety and correct task execution. Safe trajectories are obtained by preventing possible collisions with the human co-worker and by reducing the robot's velocity in case of close interaction with the human(s). In extreme cases, collision detection and reaction mechanisms are exploited to reduce the severity of possible injuries. The correct execution of a cooperative task requires the ability of the robot to adapt its plan to changes in the environment and to human intentions and needs. Top left image downloaded from http://www.iran-daily.com/News/121398.html in September 2017.

den and unexpected changes in the scene. In extreme cases, collisions with the humans cannot be avoided, and the safety of the operators has to be ensured by adopting collision detection and reaction strategies [58,60]. The dangerousness of a robot trajectory for the humans can be measured using various danger indexes [68, 84, 87]. This information is then incorporated in the robot control strategy to produce safe trajectories. In general, however, it is beneficial to avoid dangerous situations by generating collision-free paths for the robot.

The approaches for generating collision-free paths can be divided into two categories: path planning approaches and reactive motion generation approaches. The former category includes global approaches which are able to find the shortest collision-free path even in very complex scenarios with multi degrees-of-freedom robots [37]. Geometric planners [38, 144, 145] belong to the first category. Geometric planners formulate the human-robot interaction as an optimization problem, where the cost to minimize depends on various aspects like human and robot kinematics, objects in the scene, human needs and preferences. A grid based representation of the world, which also includes the human and the robot, is usually adopted and the robotic task is computed by finding the minimum-cost path in the grid [145]. Geometric planners are able to generate safe (collision-free), feasible (robot physical limitations), and socially acceptable (human preferences and needs) plans. However, despite the possibility to parallelize the algorithms in order to reduce the computation time [153], the computation time is still too large to apply these algorithms on-line.

Reactive motion generation approaches are local algorithms which change the robot path in real-time. Many approaches for reactive collision avoidance make use of a virtual potential field that attracts the robot towards a goal position while repelling it from eventual obstacles. The idea of potential fields was proposed in the pioneering work of Khatib [79]. A known drawback of the potential field approach is that the motion can stop in a local minima even if a collision-free path to the goal exists. A solution to skip the local minima is proposed in [23] by combining the benefits of the path planning algorithms with the velocity of the reactive techniques. In this method, the initial elastic band is computed off-line using a path planning algorithm, which results in a collision-free path. In the presence of obstacle, the band is deformed by applying repulsive forces. However, if the path being executed gets infeasible because of an obstacle coming into its way, the reshaping method cannot be applied any more, and an off-line replanning step is needed [167]. Other researchers propose to avoid local minima by modifying the trajectory of a particular dynamical system. For example, in [62, 114] an additive term is applied to a discrete dynamic movement primitive (DMP) [65] in order to deform the trajectory and avoid a point obstacle. The global stability of the modified system is proven with static obstacles using the Lyapunov theorem (see Appendix A.1). In [61] a potential field is applied to a second order system with varying stiffness that generates a smooth collision-free path. A combination of potential fields and circular fields is proposed in [59]. Several experiments show the convergence properties to the goal of this approach, also in very complex scenarios.

Aforementioned approaches work only with a specific dynamical system, namely a second-order dynamical system, reducing the possibility of encoding more complex, non-linear skills. A technique to modulate a generic dynamical system (DS) is proposed in [77]. Given the analytical representations of the obstacles surface, a modulation matrix is computed that locally deforms the original system. This approach can be applied on a variety of DS (both stable and unstable) and it guarantees the impenetrability of convex obstacles. Moreover, the approach in [77] does not modify the equilibrium points of the modulated system. The modulation technique in [77] is extended in [136, 137] to consider the more realistic scenario where the robot's is monitored via a RGB-D camera and the objects are represented via point clouds.

Any approach for reactive collision avoidance requires at least the distance between the robot and eventual obstacles in the scene. Hence, the robot's workspace has to be continuously monitored with exteroceptive sensors. Visual sensors are probably the most widely used exteroceptive sensors in robotic applications, and the spread of RGB-D sensors made readily available a representation of workspace as a depth image. An usual approach to use the spatial scene information for distance evaluation algorithms is to transform the data given by the visual sensors into 3D point clouds [12, 112]. However, operating directly on the depth image provided by a depth sensor increases the performance, since no transformation is needed [51, 52]. The algorithm in [51, 52] approximates the robot's body with a set of spheres. Alternatively, one can compute the distance of the obstacles form the surface of each link using a triangular mesh model, resulting in a more accurate distance estimation. Using mesh models is definitely more accurate, but computationally expensive. In order to speed up the minimum distance search, one can represent the depth map from the sensor (with the robot removed) and the virtual depth map generated from the mesh model in a structured way, such as bounding trees [122] or kd-trees [11]. The tree representation of the virtual depth can be precomputed off-line and updated at each iteration using the measured joint angles. On the contrary, the tree representation of the real depth has to be recomputed each time new sensor data become available. Computing the tree representations (bounding or kd) of a typical depth image (640×480 pixels) does not match real-time requirements [11, 122]. The time limitations of tree representations are alleviated in [73] by using a voxel grid. A voxel grid is an approximated representation of a point cloud where multiple points are mapped into a voxel (cube) of fixed size. The voxel grid creation can be easily parallelized and the distance between voxel grids can be efficiently computed (in parallel) by considering the distance between voxel's centroids. The fast distance evaluation in the depth space (fDED) approach presented in [134], instead, reduces the computation time by creating a lattice of robot and scene points, and by iteratively refining the distance estimation. As discussed in Section 5.1, fast distance computation, dynamical systems modulation, and dynamical system rescaling can be effectively combined to realize a reactive planning approach that takes into account both the human safety and the proper task execution.

2.3.2. Execution of structured tasks

In order to execute a structured task, or, in general, a task consisting of multiple actions, one has to solve the problem of deciding the next action to execute. In other words, an algorithm is required to generate the task plan. As detailed in Section 5.2, this work exploits an attentional system, a set of weights (emphasis values), and a set of logic rules (pre- and post-conditions) to determine the most emphasized robot's behavior and decide the next action to execute. The presented mechanism takes into account contextual information, like objects in the scene, human commands, and human intervention, allowing a cooperative execution of the learned task. It is worth noticing that the attentional system does not generate the entire task plan in one shot, but it periodically checks the current state and context to determine the next action. In this way, the task can be quickly replanned in case of unexpected changes in the workspace, e.g., if a human co-worker takes an object from the scene.

Instead of using semantic rules and weights, the problem of deciding the next motion to execute can also be treated as a classification problem. For instance, the approach in [115] uses nearest neighbor classification to determine the next action to execute. In [99] a graph is used to represent transitions between elementary motions. A classifier associated with each node in the graph determines when a transition occurs, i.e., when a motion is finished and the robot can execute the next one. These approaches permit to learn and reproduce complex robotic tasks from human demonstrations. However, they do not consider the possibility of executing the learned tasks in cooperation with the human. Other work in the context of human-robot collaboration are proposed in [81,97]. In this case, collaborative activities are recognized in order to infer the future human actions. Human action anticipation is used by the robot to generate the right response to the human behavior [81], enhancing the human–robot collaboration. These approaches consider the robot as an assistant, which is unable to autonomously execute the task. In contrast, the approach presented in this work permits to execute a learned task autonomously or in cooperation with a human co-worker.

Finally, it is interesting to point-out that the attentional system generates only a set of action labels with associated objects. In other words, the attentional system only considers an abstract representation of the task at hand, without considering how the commanded action is effectively executed on the real robot. This is a common choice in many approaches that exploits high-level task representations [26, 27, 124, 150, 169], which usually rely on a set of pre-programmed motion primitives to generate the motor commands. In contrast, the approach used in this work allows to learn both the task structure and the associated motion primitives from demonstration. On the other hand, the presented geometric planners [38, 144, 145] directly generate feasible robot trajectories, but at a significantly higher computational cost. The solution adopted in this work, instead, focuses on high-level task generation and relies on a low-level system capable of generating safe and feasible robot's trajectories.

2.4. Contributions

The thesis presents a **framework for intuitive transfer of structured tasks from a human teacher to a robotic learner** (see Figure 1.1). Roughly specking, a structured task consists of several actions and objects to manipulate. For some actions the order of execution is important, for other actions is arbitrary. A structured task can always be hierarchically decomposed in different subtasks involving multiple primitive actions and manipulated objects. The process to acquire a structured task has three main aspects, namely the task representation, learning, and execution. For all the three aspects, the thesis first focuses on basic skills (single movements) and then extends the methodologies to structured tasks.

Concerning the representation of basic skills, this work **identifies the limitations of Cartesian representations and presents effective solutions**. In particular, Cartesian trajectories are not compact representations and are strongly dependent on the reference frame used to represent the motion. These limitations reduce the generalization capabilities of Cartesian representations and increase memory requirements. As a possible solution, the thesis investigates the adoption of **representations invariant to affine transformations**, as well as of **dynamical system based representations**. Due to the proved convergence to a given target, dynamical systems are well-suited to represent point-to-point motions, which are the elementary actions constituting a structured task. Dynamical systems are effective in representing basic skills constituted of single motions. Beyond this representation, rooted trees are identified as the better-suited topological structure to symbolically represent structured tasks.

Regarding the learning of robotic skills, this work presents a **unified approach for incremental learning of end-effector and null-space primitives through physical guidance**. The approach leverages an incremental learning algorithm and a customized kinematic control to permit a natural refinement of end-effector and null-space movements. Moreover, the thesis presents a **framework for intuitive transfer of structured tasks**. The framework combines action segmentation and labeling, programming by demonstration, and cognitive control mechanisms to allow natural teaching and flexible/collaborative execution of structured tasks.

The framework employed for intuitive task transfer is also applied to **generate task plans for the robot and to supervise the task execution**. In particular, contextual information—objects in the scene, explicit human commands, and human intervention—are used to rapidly replan the task. Commanded actions are executed on real robots in realistic environments. Dynamical system properties and real-time robot–obstacle distance estimation are leveraged to reshape the robot trajectories, avoiding possible collisions with unforeseen obstacles (including humans) without compromising the task execution.

CHAPTER 3

Representation of robotic tasks

This chapter discusses the problem of representing a robotic skill in a way that is useful to generate the motor commands for the robot at run time. The first section presents approaches to encode the skill into a stable dynamical system, that is an effective solution to represent point-to-point motions. The second section focuses on invariant representations of rigid body motion trajectories, and on their applicability to motion recognition and reproduction problems. The last section formally defines a structured task and discusses how structured tasks are represented in this work.

3.1. Robotic skills representation using stable dynamical systems

Stable dynamical systems (DS) are able to generate motion trajectories that are guaranteed to converge at a specified target. Moreover, stable DS are flexible enough to accurately represent complicated motions [18, 78, 93], and they are able to react, in real-time, to external perturbations like changes in the desired position or unforeseen obstacles [62, 77, 136, 137]. All those features make stable DS well-suited to represent robot's discrete movements, also called point-to-point motions, which are spatial motions ending at a specified target.

In general, a DS is a non-linear mapping between a state variable $\boldsymbol{x}(t) \in \mathbb{R}^n$ and its time derivative $\dot{\boldsymbol{x}}(t) \in \mathbb{R}^n$ in the form (see Appendix A.1)

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)), \tag{3.1}$$

where $\boldsymbol{f}(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is a non-linear and smooth (continuous and continuously differentiable) function. The state variable $\boldsymbol{x}(t)$ usually represents the robot position (first-order DS) or position and velocity (second-order DS) in joint or Cartesian space. Stable DS can be learned from human demonstrations by exploiting non-linear regression techniques (see Appendix A.3), obtaining the system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{\theta}), \tag{3.2}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a vector of parameters depending on the adopted regression algorithm. Note that, depending on the definition of the state variable $\boldsymbol{x}(t)$, the system in (3.2) can be either a first- or a second-order dynamical system.

3.1.1. Learning stable motions from a single demonstration

Originally proposed by Ijspeert et al., the dynamic movement primitives (DMPs) framework represents one of the first attempts to encode a single task demonstration into a stable dynamical system [65–67]. The original formulation uses a separate DMP for each degrees-of-freedom, and it suffers form two main limitations: *1)* it is not possible to represent motions that have the same start and goal positions, and *2)* large changes in the initial or goal positions may cause a magnification of the trajectory resulting in high accelerations of the robot. Those limitations are overcome by the DMP formulation proposed in [62, 114] and summarized as follows.

DMPs encode a motion primitive into a second order, non-linear dynamical system, composed by a linear (spring–damper) dynamics and a non-linear forcing term. A DMP is defined as

$$\tau\dot{\boldsymbol{x}} = \boldsymbol{v}, \tag{3.3a}$$
$$\tau\dot{\boldsymbol{v}} = \boldsymbol{K}(\boldsymbol{g}-\boldsymbol{p}) - \boldsymbol{D}\boldsymbol{v} - \boldsymbol{K}(\boldsymbol{g}-\boldsymbol{x}_0)s + \boldsymbol{K}\boldsymbol{f}(s), \tag{3.3b}$$
$$\tau\dot{s} = -\gamma s, \tag{3.3c}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the robot position in joint or Cartesian space, $\boldsymbol{v} \in \mathbb{R}^n$ and $\dot{\boldsymbol{v}} \in \mathbb{R}^n$ are the robot velocity and acceleration respectively, $\boldsymbol{g} \in \mathbb{R}^n$ the desired (goal) position, $\boldsymbol{x}_0$ is the initial position of the robot, $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ are positive definite gain matrices. The gain τ is a temporal scaling factor used to alter the duration of the motion. s is a clock signal initialized to $s_0 = 1$, and exponentially decaying to $s \to 0$ for any value of the gain $\gamma > 0$. The value of γ affects the convergence time of s. In particular, being (3.3c) a linear system, it holds that $s \approx 0$ after $5/\gamma$ seconds. The nonlinear forcing term $\boldsymbol{f}(\cdot) \in \mathbb{R}^n$ reshapes the linear dynamics to follow the demonstrated trajectory. The forcing term is defined as

$$\boldsymbol{f}(s) = \frac{\sum_{k=1}^{K} \phi_k(s)\boldsymbol{w}_k}{\sum_{k=1}^{K} \phi_k(s)} s, \tag{3.4}$$

which underlines that $\boldsymbol{f}(s)$ is deactivated by the clock signal, i.e. $\boldsymbol{f}(s) \to \boldsymbol{0}$ for $s \to 0$. This guarantees the convergence to the desired goal $\boldsymbol{g}$. The functions $\phi_k(s)$ in (3.4) are Gaussian basis functions centered at c_k and with spread h_k, i.e. $\phi_k(s) = \exp(-h_k(s-c_k)^2)$, while $\boldsymbol{w}_k$ are adjustable weights. The centers c_k and the spreads h_k are manually specified, while the weights $\boldsymbol{w}_k$ are learned from a single demonstration using weighted least squares (see Appendix A.3.1).

3.1.2. Gaussian mixture regression based dynamical systems

Gaussian mixture models/regression (GMM/GMR) can be exploited to represent $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{\theta})$ in (3.2) as a finite mixture of Gaussian components. As detailed in Appendix A.3, GMM parameterize a non-linear function with the vector $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$, where π_k are the priors, $\boldsymbol{\mu}_k$ the means, and $\boldsymbol{\Sigma}_k$ the covariance matrices of each Gaussian components. Given N task demonstrations $\mathcal{D} = \{\boldsymbol{x}_{t,i}, \dot{\boldsymbol{x}}_{t,i}\}_{t=1,i=1}^{T,N}$, where $\boldsymbol{x}_{t,i} \in \mathbb{R}^n$ is the position and $\dot{\boldsymbol{x}}_{t,i} \in \mathbb{R}^n$ the velocity of the robot in joint or Cartesian space, the parameters vector is learned through the Expectation–Maximization algorithm [39]. Means and covariance matrices are defined as

$$\boldsymbol{\mu}_k = \begin{bmatrix} {}^{x}\boldsymbol{\mu}_k \\ {}^{\dot{x}}\boldsymbol{\mu}_k \end{bmatrix}, \quad \boldsymbol{\Sigma}_k = \begin{bmatrix} {}^{x}\boldsymbol{\Sigma}_k & {}^{x\dot{x}}\boldsymbol{\Sigma}_k \\ {}^{\dot{x}x}\boldsymbol{\Sigma}_k & {}^{\dot{x}}\boldsymbol{\Sigma}_k \end{bmatrix}. \tag{3.5}$$

Having learned the parameters $\boldsymbol{\theta}$ from demonstration, GMR is exploited to generate the robot's trajectory. In particular, a probability density function $p(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t); \boldsymbol{\theta})$, in the form of a mixture of

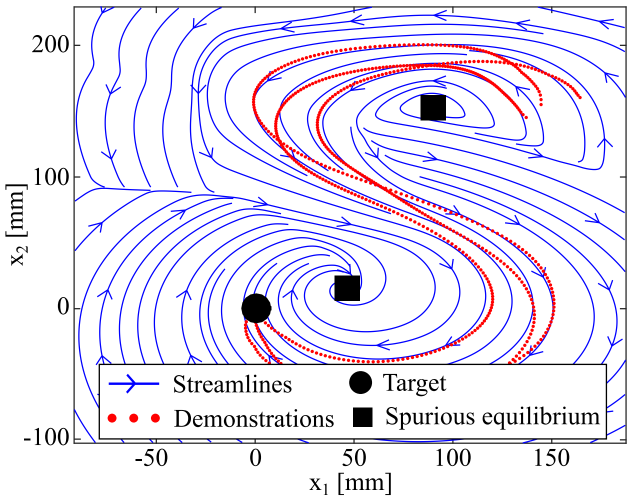


Figure 3.1.: A point-to-point motion learned by GMR. The standard GMR approach generates spurious equilibrium points.

Gaussian components (see Equation (A.21)), is associated to each point in the state space. Taking the mean of the posterior probability $p(\dot{\boldsymbol{x}}(t)|\boldsymbol{x}(t),\boldsymbol{\theta})$ as an estimation of $\boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{\theta})$ yields [76]

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{\theta}) = \sum_{k=1}^{K} h_k(\boldsymbol{x}(t))(\boldsymbol{A}_k\boldsymbol{x}(t) + \boldsymbol{b}_k), \tag{3.6}$$

where

$$\begin{aligned} \boldsymbol{A}_k &= {}^{\dot{x}x}\boldsymbol{\Sigma}_k({}^{x}\boldsymbol{\Sigma}_k)^{-1}, \\ \boldsymbol{b}_k &= {}^{\dot{x}}\boldsymbol{\mu}_k - \boldsymbol{A}_k{}^{x}\boldsymbol{\mu}_k, \\ h_k(\boldsymbol{x}) &= \frac{\pi_k\mathcal{N}(\boldsymbol{x}(t)|{}^{x}\boldsymbol{\mu}_k,{}^{x}\boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K}\pi_i\mathcal{N}(\boldsymbol{x}(t)|{}^{x}\boldsymbol{\mu}_i,{}^{x}\boldsymbol{\Sigma}_i)}. \end{aligned} \tag{3.7}$$

From Equation (3.6), it is clear that GMR represents a non-linear DS as a non-linear summation of linear dynamical systems.

A known problem (see, for example, [18,76]) of the GMR based DS representation is that the generated trajectory does not always converge to a unique equilibrium point. This limitation is also shown in Figure 3.1. An effective solution to this problem is presented as follows.

3.1.3. Contracting Gaussian mixture regression

The Contracting Gaussian mixture regression (C-GMR) is an approach to stabilize, at run time, a dynamical system represented by GMR, forcing the robot's trajectory to reach the specified target. The approach exploits Partial Contraction theory (see Appendix A.1.2) to derive sufficient stability conditions for the DS in (3.6), and to automatically compute a stabilizing control input. An activation function is also used to guarantee an accurate reproduction of the demonstrated motions while preserving the convergence properties.

Contraction Analysis

In order to show that (3.6) is contracting, one has toshow that the Jacobian matrix $\partial\boldsymbol{f}(\cdot)/\partial\boldsymbol{x}$ is uniformly negative definite (see Definition 3 in Table A.1). Computing the Jacobian is not trivial, due to the non-linear term $\partial h_k(\boldsymbol{x})/\partial \boldsymbol{x}$ in (3.6). Instead of computing the Jacobian, Partial Contraction theory (see [159] and Appendix A.1.2) is exploited to significantly simplify the problem. Note that, in order to simplify the notation, the time dependency of the state variables is omitted in the rest of the section.

According to the Partial Contraction theory formulation, an auxiliary DS is defined as

$$\dot{\boldsymbol{y}} = \sum_{k=1}^{K} h_k(\boldsymbol{x})(\boldsymbol{A}_k \boldsymbol{y} + \boldsymbol{b}_k), \tag{3.8}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ is the auxiliary state variable, and $\dot{\boldsymbol{y}}$ is the time derivative of $\boldsymbol{y}$. Notably, the Jacobian of the auxiliary system has a simple expression, i.e. $\boldsymbol{J}_a = \sum_{k=1}^{K} h_k(\boldsymbol{x})\boldsymbol{A}_k$. It is known that, if the auxiliary system in (3.8) is globally contracting and it has an equilibrium point $\boldsymbol{x}^*$, then the original system in (3.6) globally exponentially converges to $\boldsymbol{x}^*$ [159]. Theorem 1 gives sufficient conditions for the global exponential stability of the DS in (3.6).

Theorem 1. *The dynamical system in (3.6) globally exponentially converges to a unique equilibrium point $\boldsymbol{x}^*$ if, for all $k = 1, \dots, K$, it holds that:*

- $\boldsymbol{b}_k = -\boldsymbol{A}_k \boldsymbol{x}^*$ (3.9)
- $\exists \mu(\cdot)$ *such that* $\mu(\boldsymbol{A}_k) < 0.$ (3.10)

Proof. Condition (3.9) guarantees that $\boldsymbol{x}^*$ is an equilibrium point of the the auxiliary system (3.8). Indeed, by substituting (3.9) into (3.8), it is easy to obtain that $\sum_{k=1}^{K} h_k(\boldsymbol{x})\boldsymbol{A}_k(\boldsymbol{y} - \boldsymbol{x}^*) = \boldsymbol{0}$ if $\boldsymbol{y} = \boldsymbol{x}^*$. Condition (3.10) guarantees that the auxiliary system is globally contracting. Indeed, recalling that $\mu(\alpha \boldsymbol{A}) = \alpha\mu(\boldsymbol{A})$ (see Definition 2 in Table A.1), it holds that

$$\mu(\boldsymbol{J}_a) = \mu\left(\sum_{k=1}^{K} h_k(\boldsymbol{x})\boldsymbol{A}_k\right) = \sum_{k=1}^{K} h_k(\boldsymbol{x})\mu(\boldsymbol{A}_k).$$

Considering condition (3.10) and the property $0 \leq h_x \leq 1$ from (3.7), it is easy to verify that $\mu(\boldsymbol{J}_a) = \sum_{k=1}^{K} h_k(\boldsymbol{x})\mu(\boldsymbol{A}_k) < 0$ for all $\boldsymbol{y} \in \mathbb{R}^n$. Recalling that an autonomous and globally contracting DS has a unique equilibrium at $\boldsymbol{x}^*$ [95], it is possible to conclude that the DS (3.6) globally exponentially converges to $\boldsymbol{x}^*$. □

A result similar to that in Theorem 1 can be obtained by analyzing the Lyapunov stability of the DS in (3.6) (see [76]). The difference is that Lyapunov theory uses the matrix measure associated to the Euclidean norm, while any matrix measure can be used to verify the condition (3.10). This advantage of Contraction theory is exploited to automatically compute a stabilizing control input.

Global exponential stabilizer

A possible way to stabilize, at run time, the DS in (3.6) consists in applying a control input

$$\dot{\boldsymbol{x}} = \underbrace{\sum_{k=1}^{K} h_k(\boldsymbol{x})\,(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{b}_k)}_{System} + \underbrace{\sum_{k=1}^{K} h_k(\boldsymbol{x})\,(\boldsymbol{U}_k\boldsymbol{x} - \boldsymbol{A}_k\boldsymbol{x}^\star)}_{Control} \tag{3.11}$$

The global exponential convergence of the controlled system in (3.11) is proved by considering the auxiliary DS $\dot{\boldsymbol{y}} = \sum_{k=1}^{K} h_k(\boldsymbol{x})(\boldsymbol{A}_k\boldsymbol{y} + \boldsymbol{b}_k) + \sum_{k=1}^{K} h_k(\boldsymbol{x})(\boldsymbol{U}_k\boldsymbol{y} - \boldsymbol{A}_k\boldsymbol{x}^*)$, and by checking if the conditions in Theorem 1 are satisfied. The auxiliary DS can be re-written as

$$\dot{\boldsymbol{y}} = \sum_{k=1}^{K} h_k(\boldsymbol{x})\,(\boldsymbol{A}_k + \boldsymbol{U}_k)\,(\boldsymbol{y} - \boldsymbol{x}^*). \tag{3.12}$$

Algorithm 1 Find a matrix $\boldsymbol{U}$ such that $\mu_1(\boldsymbol{A}+\boldsymbol{U}) < 0$

```
Require: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $p > 1$, $n$        // dimension of state space
  $\boldsymbol{U} \leftarrow$ Matrix of zeros in $\mathbb{R}^{n\times n}$
  for $d = 1$ to $n$ do
    $s = \sum_{i\neq d} |a_{di}|$
    if $a_{dd} > 0$ and $a_{dd} < s$ then
      $u_{dd} \leftarrow -s - pa_{dd}$
    else if $a_{dd} > 0$ and $a_{dd} > s$ then
      $u_{dd} \leftarrow -2a_{dd}$
    else if $a_{dd} < 0$ and $|a_{dd}| < s$ then
      $u_{dd} \leftarrow -s$
    else
      $u_{dd} \leftarrow 0$                        // no need to modify $a_{dd}$
    end if
  end for
  return $\boldsymbol{U}$
```

It is easy to show that the controlled DS in (3.12) has an equilibrium at $\boldsymbol{x}^*$. Indeed, the term $-\sum_{k=1}^{K} h_k(\boldsymbol{x})\boldsymbol{A}_k\boldsymbol{x}^*)$ of the control input in (3.11) is used to satisfy condition (3.9) in Theorem 1. Moreover, by looking at the controlled DS in (3.12), it is clear that the control matrices $\boldsymbol{U}_k$ have to guarantee that $\mu(\boldsymbol{A}_k+\boldsymbol{U}_k) < 0,\ k = 1,..,K$ (condition (3.10) in Theorem 1).

The control matrices $\boldsymbol{U}_k$ can be automatically computed by using the automatic procedure in Algorithm 1, originally proposed in [18]. The algorithm exploits the $\mu_1(\cdot)$ matrix measure associated to the l_1-norm, which is defined as

$$\mu_1(\boldsymbol{C}) \triangleq \max_j \left(c_{jj} + \sum_{i\neq j} |c_{ij}| \right),$$

where c_{ij} indicates the element at row i and column j of $\boldsymbol{C}$. From the definition of $\mu_1(\cdot)$ matrix measure, it derives that $\mu_1(\boldsymbol{C}) < 0$ if the matrix $\boldsymbol{C}$ has negative diagonal elements $c_{ii} < 0,\ i = 1,\dots,n$ and it is diagonally dominant $|c_{ii}| > \sum_{j=1}^{n} |c_{ij}|,\ i = 1,\dots,n$. Given a square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, Algorithm 1 finds a diagonal matrix $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ such that $\boldsymbol{C} = \boldsymbol{A}+\boldsymbol{U}$ has negative diagonal elements and it is diagonally dominant, guaranteeing that $\mu_1(C) < 0$. In details, Algorithm 1 inspects each row of the matrix $\boldsymbol{A}$ and it computes, for each row, the sum of the off-diagonal elements $s = \sum_{i\neq d} |a_{di}|$. The algorithm then considers four different cases. In the first case, the d-th diagonal element of $\boldsymbol{A}$ is positive ($a_{dd} > 0$) and $a_{dd} < c$. In this case, $u_{dd} = -s - pa_{dd}$. In the second case, $a_{dd} > 0$ and $a_{dd} < s$, and then $u_{dd} = -2a_{dd}$. In the third case, $a_{dd} < 0$ and $|a_{dd}| < s$. This implies that $u_{dd} = -s$. In all the other cases $\boldsymbol{A}$ has already a negative matrix measure. The described approach to select the values of each u_{dd} guarantees that $a_{dd} + u_{dd} < 0$ and that $|a_{dd} + u_{dd}| > c$ for $d = 1,\dots,n$, i.e. that $\mu_1(\boldsymbol{A}+\boldsymbol{U}) < 0$.

The gains of the stabilizing control are automatically computed given the matrices $\boldsymbol{A}_k$, which only depends on the learned GMM parameters (see Equation (3.7)), and the GMR based DS is stabilized at run time. In contrast to the approaches in [76, 78, 93], the stabilization of the DS does not require additional learning steps. Qualitative results of this stabilization approach are shown in Figure 3.2. It is worth noticing that the control input significantly affects the reproduced trajectories, preventing an accurate reproduction of the demonstrations. This is because the controller imposes an exponentially converging dynamics to the DS in (3.11), which contradicts the demonstrations. In formal terms, $\exists\beta > 0$ such that $\|\boldsymbol{x}(t) - \boldsymbol{x}^*\|_1 \leq \beta\|\boldsymbol{x}(0) - \boldsymbol{x}^*\|_1 e^{-ct}$, where

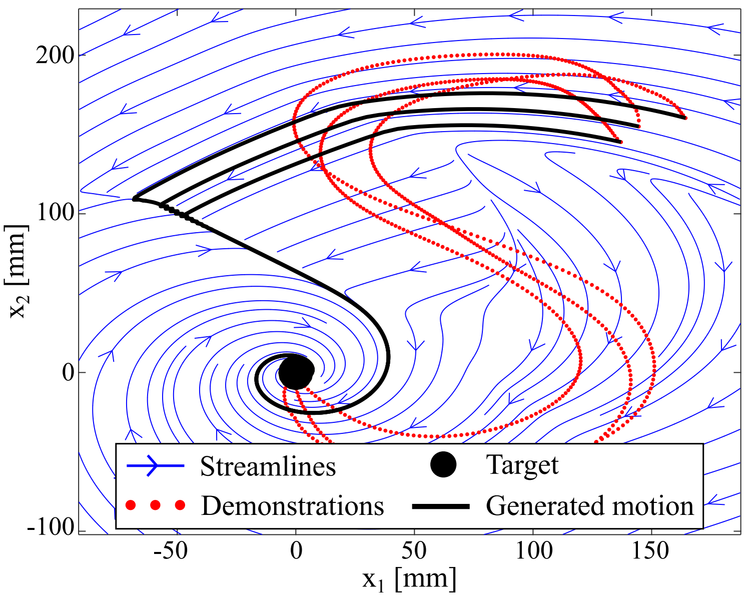


Figure 3.2.: The controlled system (3.11) generates an exponentially converging trajectory. The trajectory converges to $\boldsymbol{x}^* = [0,0]^{\mathrm{T}}$, but with a significant reproduction error.

$\|\cdot\|_1$ is the l_1-norm and $\mu_1(\boldsymbol{A}_k + \boldsymbol{U}_k) \leq -c < 0, \forall k$. This problem is alleviated by introducing an activation signal, which helps to balance between accurate motion reproduction and stable motion generation.

Accurate reproduction of stable motions

The control law in (3.11) is modified in order to improve the reproduction accuracy and to preserve the stability properties of the controlled system. The new controlled system is given by

$$\dot{\boldsymbol{x}} = \underbrace{\sum_{k=1}^{K} h_k(\boldsymbol{x})\left(\boldsymbol{A}_k\boldsymbol{x} + \boldsymbol{b}_k\right)}_{System} + \underbrace{\omega(\boldsymbol{x},t)\left(\sum_{k=1}^{K} h_k(\boldsymbol{x})\left(\boldsymbol{U}_k\boldsymbol{x} - \boldsymbol{b}_k\right)\right)}_{Control}, \tag{3.13}$$

where the so-called *activation function* $\omega(\boldsymbol{x},t) \in \mathbb{R}$ in (3.13) is used to balance between accurate reproduction and stability. In particular, the controller is active for $\omega(\boldsymbol{x},t) = 1$ and the controlled DS exponentially converges towards the target $\boldsymbol{x}^*$. The controller is deactivated for $\omega(\boldsymbol{x},t) = 0$, meaning that the demonstrations are accurately reproduced. The activation function follows the linear dynamics

$$\begin{cases} \dot{\omega}(\boldsymbol{x},t) = -\gamma(\omega(\boldsymbol{x},t) - c(\boldsymbol{x})) & t < t_{max} \\ \omega(\boldsymbol{x},t) = 1 & t \geq t_{max} \end{cases}, \tag{3.14}$$

that is stable for each value of $\gamma > 0$. The value of γ is set considering that, in practice, $\omega(\boldsymbol{x},t) = c(\boldsymbol{x})$ after $(5/\gamma)$ s. The scalar function $c(\boldsymbol{x})$ plays the role of an equilibrium point for the system in (3.14), i.e. $\omega(\boldsymbol{x},t) \to c(\boldsymbol{x})$ for $t \to +\infty$. The value of $c(\boldsymbol{x})$ is 0 when the controller has to be deactivated, and it switches to $c(\boldsymbol{x}) = 1$ to activate the controller.

The control structure in (3.13) and (3.14) allows smooth activation/deactivation of the stabilizing control input. This property is exploited to achieve an accurate reproduction of the demonstrations without compromising the stability, following the idea that the motion is accurate inside a certain area around the demonstrations. To this end, the state space is divided into three regions:

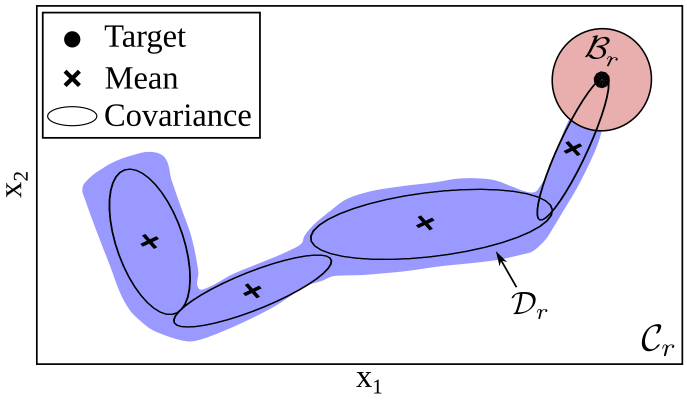


Figure 3.3.: The state space partitioned into three regions. $\mathcal{D}_r$ is the demonstration area, while the union $\mathcal{C}_r \cup \mathcal{B}_r$ is a contracting region.

an area around the demonstrations indicated with $\mathcal{D}_r$, a contraction region called $\mathcal{C}_r$, and a ball of radius r centered at the target $\mathcal{B}_r$. The described partitioning is shown in Figure 3.3. The value of $c(\boldsymbol{x})$ is then chosen as $c(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{D}_r$ (control deactivated), and $c(\boldsymbol{x}) = 1$ if $\boldsymbol{x} \in \mathcal{C}_r \cup \mathcal{B}_r$ (control active). Trajectories that start in $\mathcal{C}_r \cup \mathcal{B}_r$ follow an exponentially convergent dynamics unless they reach the equilibrium $\boldsymbol{x}^*$, or they enter the demonstration area $\mathcal{D}_r$. The controller guarantees the contraction in $\mathcal{C}_r \cup \mathcal{B}_r$, so this region is named a contracting region. The control action is also activated in case the DS has not converged within t_{max} seconds. This prevents the generated trajectories from converging to eventual spurious attractors located inside the demonstration area. It is worth noticing that, in general, a GMR with a proper number of components does not generate spurious attractors within $\mathcal{D}_r$. Indeed, the GMR generates a velocity close to the demonstrated one(s), which is zero only at the target. This means that the velocity retrieved with GMR drops to zero only close to the given target. This result is empirically shown in Section 3.1.4.

As discussed, the presented control structure permits an accurate reproduction inside the demonstration area. Theorem 2 provides conditions for the global asymptotic stability of the controlled DS in (3.13).

Theorem 2. *The dynamical system in (3.13), with activation function (3.14), globally asymptotically converges to $\boldsymbol{x}^*$*

Proof. Recall that, in order to show that a system is asymptotically stable, one has to analyze the convergence properties of the system for $t \rightarrow +\infty$. Given the dynamics in (3.13), it is easy to notice that the systems in (3.13) and (3.11) coincide for $\omega(x,t) = 1$. Now, the DS in (3.11) is proved to be globally exponentially stable. Considering that $\omega(x,t) = 1$ for $t \geq t_{max}$, it is possible to conclude that the DS in (3.13) globally asymptotically converges to $\boldsymbol{x}^*$. □

In order to apply the presented controller, it is needed to build the three areas in Figure 3.3 and to determine to which region a certain state belongs to. An approach useful to construct the demonstration area $\mathcal{D}_r$ is proposed in [75] and exploited in this work. The algorithm in [75] can be executed off-line, given the training input (position) $\mathcal{D} = \{\boldsymbol{x}_{t,i}\}_{t=1,i=1}^{T,N}$. In details, each robot position in the training set is assigned to a cluster $\mathcal{D}_k$, $k = 1,\dots,K$, where K is the number of Gaussian components. A certain position $\boldsymbol{x}_{t,i}$ belongs to $\mathcal{D}_k$ if $\mathcal{N}\left(\boldsymbol{x}_{t,i}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k\right) > \mathcal{N}\left(\boldsymbol{x}_{t,i}|{}^x\boldsymbol{\mu}_j, {}^x\boldsymbol{\Sigma}_j\right)$, $\forall j \neq k$, where $\mathcal{N}(\cdot)$ is the probability that $\boldsymbol{x}_{t,i}$ is generated from the k-th component. Given the K clusters, a set of K scalars is computed as

$$\delta_k = \alpha \min_{\boldsymbol{x}_{t,i} \in \mathcal{D}_k} \mathcal{N}\left(\boldsymbol{x}_{t,i}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k\right), \tag{3.15}$$

where $0 < \alpha \leq 1$ is a constant value which defines the area of $\mathcal{D}_r$. Bigger values of α result in tighter $\mathcal{D}_r$ around the demonstrations. In all the experiments, the value $\alpha = 0.1$ is empirically chosen to prevent trajectories starting in $\mathcal{D}_r$ to exit from the demonstration area. The scalar δ_k defined by (3.15) represents the probability of the point in $\mathcal{D}_k$ located at the maximum distance from ${}^x\boldsymbol{\mu}_k$, that guarantees that the region $\mathcal{D}_{tot} = \{\boldsymbol{x}_{t,i} : \mathcal{N}(\boldsymbol{x}_{t,i}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k) \geq \delta_k\}$ contains all the positions in the training set. The demonstration area $\mathcal{D}_r$ is computed as $\mathcal{D}_r = \mathcal{D}_{tot} \backslash \mathcal{B}_r$. The described algorithm to construct $\mathcal{D}_r$ is computationally efficient. Indeed, the algorithm exploits the probabilities $\mathcal{N}(\boldsymbol{x}_{t,i}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k)$ which are computed in the E–M algorithm to train the GMM and come at no extra cost. Given those probabilities, the algorithm simply finds the minimum element of K vectors, multiplies the obtained K values by a scalar α, and stores the results in δ_k, $k = 1, .., K$.

Given the demonstration area, the current state $\boldsymbol{x}$ is assigned to one of the regions at run time. In particular, the state $\boldsymbol{x}$ belongs to $\mathcal{B}_r$ if $\|\boldsymbol{x} - \boldsymbol{x}^*\| \leq r$. In case $\|\boldsymbol{x} - \boldsymbol{x}^*\| > r$, the state $\boldsymbol{x}$ belongs to $\mathcal{D}_r$ or to $\mathcal{C}_r$. The current state $\boldsymbol{x} \in \mathcal{D}_r$ if there exists a k such that $\mathcal{N}(\boldsymbol{x}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k) \geq \delta_k, k = 1, .., K$, otherwise $\boldsymbol{x} \in \mathcal{C}_r$. This classification algorithm is computationally efficient. Indeed, the probabilities $\mathcal{N}(\boldsymbol{x}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k)$, $k = 1, .., K$, are used to compute $h_k(\boldsymbol{x})$ in (3.7) and come at no extra cost. Hence, the classification algorithm only checks if the K values $\mathcal{N}(\boldsymbol{x}|{}^x\boldsymbol{\mu}_k, {}^x\boldsymbol{\Sigma}_k) \geq \delta_k$. In case $\boldsymbol{x} \in \mathcal{C}_r \cup \mathcal{B}_r$ or $t > t_{max}$, the controller is activated, which only requires to sum up the K matrices $\boldsymbol{A}_k$ and $\boldsymbol{U}_k$.

3.1.4. Results and comparisons

This section evaluates the performance of Contracting GMR (C-GMR) presented in Section 3.1.3. The evaluation considers two main aspects, namely the reproduction accuracy and the training time. C-GMR is also compared with two Lyapunov based approaches, namely the stable estimator of dynamical systems (SEDS) in [76] and the SEDSII in [78]. The goal of this comparison is to underline the basic differences between Lyapunov and Contraction based approaches in stable motion learning from multiple demonstrations. The LASA Handwritten dataset [2] is used as a benchmark for testing and comparisons. The dataset is a collection of 20 motions in 2D, collected at 50 Hz using a Tablet-PC. Each motion terminates at $\boldsymbol{x}^* = [0, 0]^{\mathrm{T}}$ and is demonstrated three times. In the experiments, a DS for each motion in the dataset is learned by GMR considering all the three demonstrations. The LASA Handwritten dataset contains motions of different complexities. In general, the number of Gaussian components needed to accurately represent a motion depends on the complexity of the motion itself. Simple motions, like linear paths, require a limited number of components, while complex non-linear motions require many components. In order to maximize the accuracy, the number of components for each motion is automatically computed using the Bayesian information criterion (BIC) [140]. Figure 3.4 shows qualitative results when C-GMR is used to stabilize a GMR based DS at run time. The sampling time used to generate the motion trajectory is set to 2 ms. C-GMR is capable to accurately learn complex point-to-point motions and to ensure global convergence towards the target.

Reproduction accuracy

The reproduction accuracy is measured by two metrics. The first metrics considers the shape of the generated trajectory. In particular, this metrics computes the area between a demonstrated trajectory $\mathcal{D}_i \in \mathcal{D}$ and the trajectory $\mathcal{T}$ generated by C-GMR, SEDS, or SEDSII. Both $\mathcal{D}_i$ and $\mathcal{T}$ start from the same point and end at the same target. For this reason, the area among $\mathcal{D}_i$ and $\mathcal{T}$ represents the deviation from the desired trajectory $\mathcal{D}_i$. Being $\mathcal{D}_i$ and $\mathcal{T}$ two trajectories of different length, multi-dimensional dynamic time warping (DTW) [132] is exploited to align the

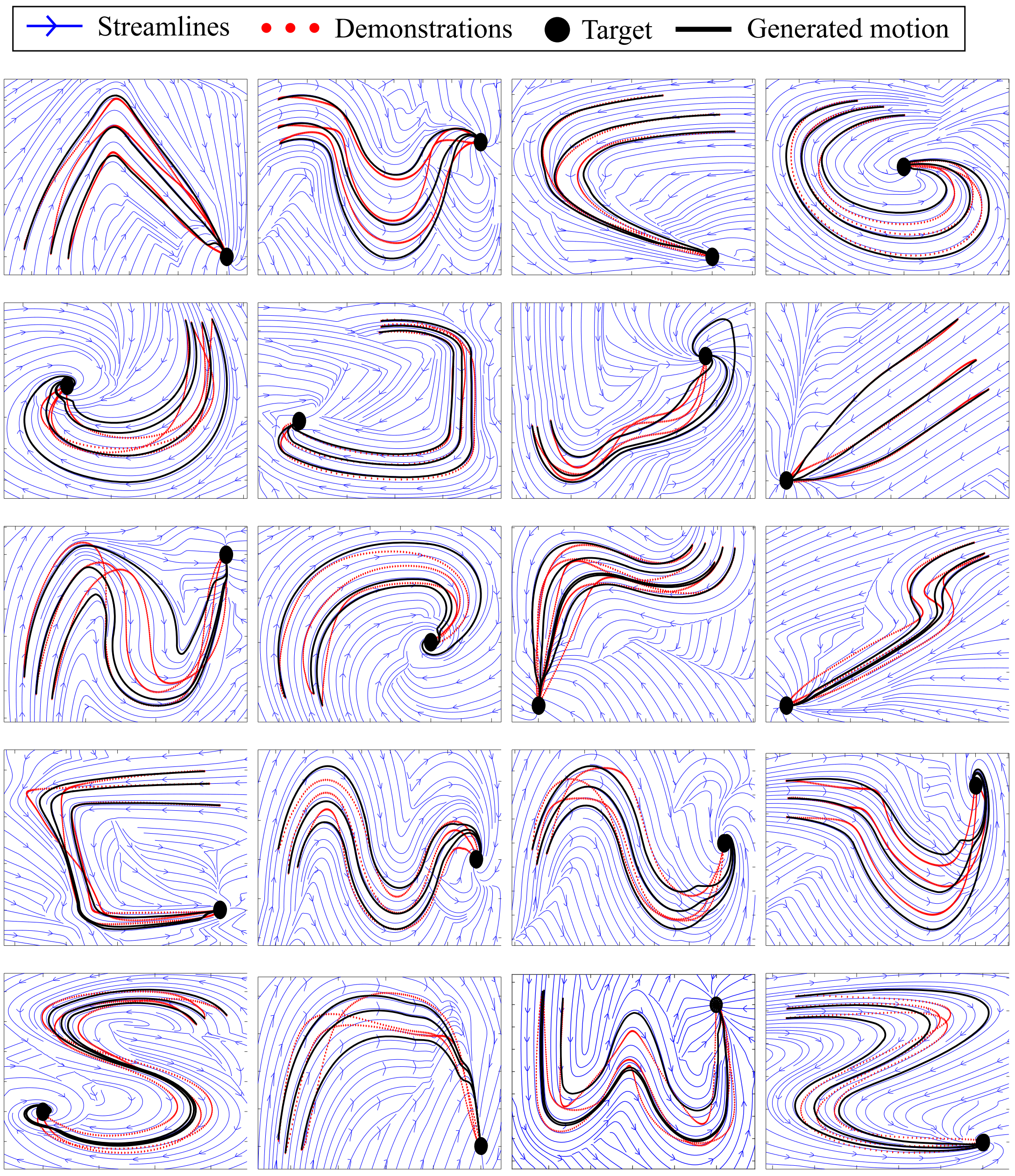


Figure 3.4.: C-GMR is capable of effectively learning and accurately reproducing stable point-to-point motions in the LASA dataset.

two trajectories and then compute the area among them. As already mentioned, the area metrics measures how well the learned DS preserves the shape of the demonstrated trajectory. In order to take into account also the kinematics of the demonstrations, the velocity root mean square error metrics [93] is used. This metrics is defined as $V_e = \sqrt{\frac{1}{NT}\sum_{t,i=1}^{T,N} \|\dot{\boldsymbol{x}}_{t,i} - \boldsymbol{f}(\boldsymbol{x}_{t,i})\|^2}$.

For the SEDSII, one has to specify the form of the control Lyapunov function (CLF) [78], which depends on the complexity of the motion. In the following experiments, two different parametrization of CLF are tested and compared. The two parameterization are referred as CLF0 and CLF3. CLF0 is defined as $CLF0 = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}^0\boldsymbol{x}$, while CLF3 is defined as $CLF3 = CLF0 + \sum_{l=1}^{3}\beta^l(\boldsymbol{x})\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}^l(\boldsymbol{x}-\boldsymbol{\mu}^l)\right)^2$. Hence, CLF0 is an ellipsoid centered around the target (symmetric

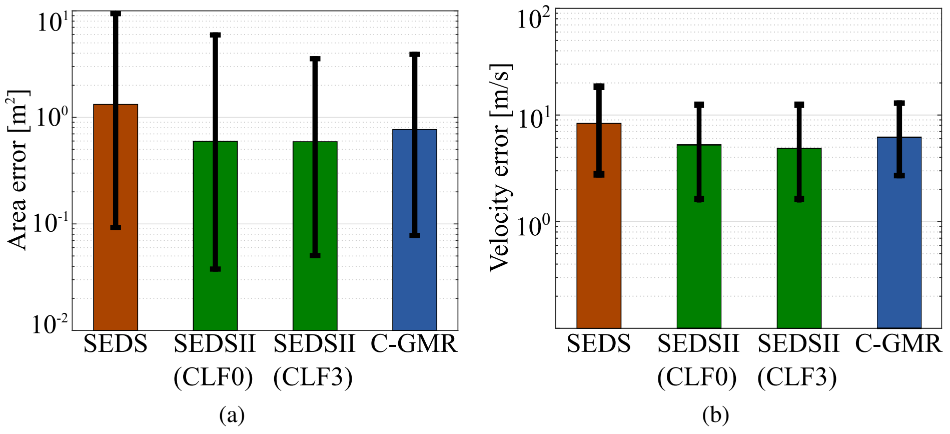


Figure 3.5.: Trajectory reproduction errors for SEDS, SEDII, and C-GMR on the LASA dataset. Black error bars are 10% and 90% quantiles of the median value.

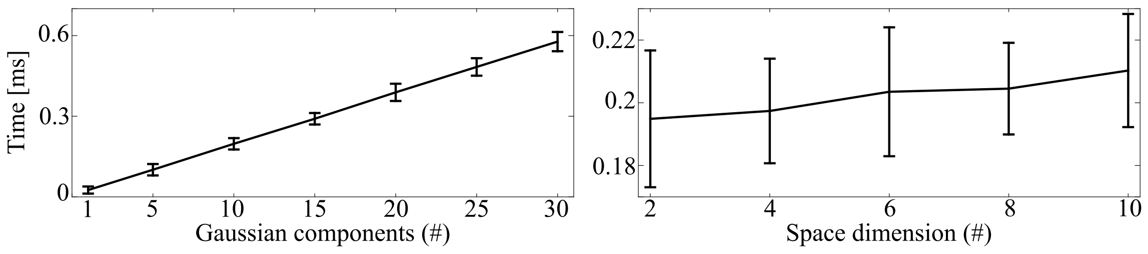
component), while CLF3 is CLF0 plus three asymmetric components. The parameters β^l, $\boldsymbol{P}^l$, and $\boldsymbol{\mu}^l$ are learned from demonstrations by solving a constrained optimization problem [78].

The learning process in GMR, SEDS, and SEDSII consists in solving optimization problems. The numerical solution of such problems is usually affected by the initial conditions of the solver. Different initial conditions may lead to different results. Hence, it is needed to perform multiple trials and report typical performance. To this end, each approach is tested ten times for each motion in the dataset. Figure 3.5 shows the overall accuracy for C-GMR, SEDS, and SEDSII (CLF0 and CLF3). Note that, for a better visualization, a logarithmic scale is used for the ordinate axis. Since the reproduction error is not normally distributed, the median value is shown in Figure 3.5 instead of the mean value. Moreover, the location of the 10% and the 90% quantiles (black error bars in Figure 3.5) indicates the maximal and minimal deviation from the median value. Results for C-GMR are obtained by setting the radius of $\mathcal{B}_r$ equal to the 15% of the distance between the starting point and the target.

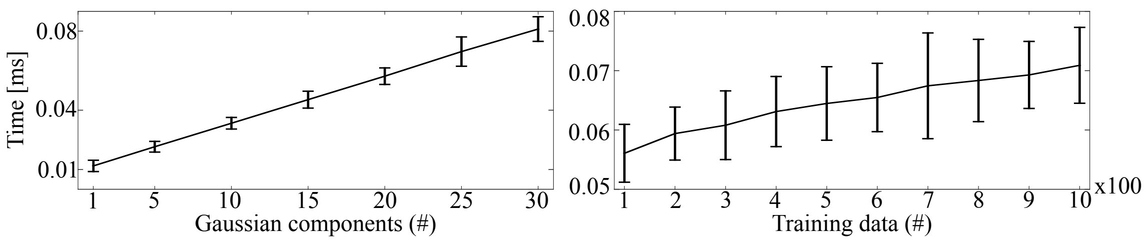
As shown in Figure 3.5, SEDS has the worst area ($1.3\,\mathrm{m}^2$) and a velocity ($8.3\,\mathrm{m/s}$) errors. As discussed in [76], SEDS derives stability constraint from the quadratic Lyapunov function $\|\boldsymbol{x}\|^2$. The results obtained with SEDS indicates that most of the motions in the dataset cannot be learned with a simple, quadratic Lyapunov function. SEDSII with CLF0 parameterization has an area error of $0.596\,\mathrm{m}^2$ and a velocity error $5.2\,\mathrm{m/s}$, while the CLF3 parameterization has an area error of $0.591\,\mathrm{m}^2$ and a velocity error $4.9\,\mathrm{m/s}$. The fact that the average accuracy of the two parameterization is similar indicates that most of the motions in the dataset can be learned with the Lyapunov function $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}^0\boldsymbol{x}$. However, SEDS with CLF3 is well-suited for all the motions in the dataset and it presents the highest accuracy. C-GMR is less accurate ($0.768\,\mathrm{m}^2$ and $6.1\,\mathrm{m/s}$) than SEDSII, but it outperforms SEDS. The reason is that C-GMR loses accuracy when the trajectory enters the ball $\mathcal{B}_r$ around the equilibrium, where the controlled DS follows an exponentially converging dynamics.

Training time

The described C-GMR is capable of stabilizing the GMR based DS at run time. The goal of this evaluation is to analyze the time performance of C-GMR, and to compare the results obtained with C-GMR, SEDS, and SEDSII. The first step is to evaluate the extra training time introduced by C-GMR. In other words, the time spent to build the demonstration area $\mathcal{D}_r$ and to compute the control



(a) Execution time of Algorithm 1, averaged over 100 executions.



(b) Time required to compute $\mathcal{D}_r$, averaged over 100 executions.

Figure 3.6.: Time required to execute Algorithm 1 (a) and to compute the demonstrations area (b).

matrices $\boldsymbol{U}_k$ (see Algorithm 1). The computation time of Algorithm 1 is a function of the number K of Gaussian components (one control matrix for each component) and the dimension n of the state space ($\boldsymbol{A}_k, \boldsymbol{U}_k \in \mathbb{R}^{n\times n}$). Figure 3.6(a) shows the average execution time of an unoptimized Matlab implementation of Algorithm 1. The left graph shows the computation time with a variable number of components ($K \in [1,30]$). The results are obtained with a constant dimension of the state space ($n = 2$ as in the LASA dataset). As expected, the computation time grows almost linearly with the number of Gaussians. The right graph shows the computation time obtained by fixing the number of Gaussian to $K = 10$, and varying the dimension of the state space ($n \in [2,10]$). Also in this case, the computation time grows linearly. The execution time is below 0.6 ms in all the considered cases. The computation time of the approach adopted to construct the demonstration area $\mathcal{D}_r$ is a function of the number of Gaussians K and the number of training points T. The approach has been implemented in Matlab, and the average execution time is shown in Figure 3.6(b). The left graph shows the computation time obtained by keeping fixed the number of training data ($T = 300$ as in the LASA dataset) and by varying number of components ($K \in [1,30]$). The right graph is the computation time obtained with $K = 10$ components and varying the number of training data ($T \in [100,1000]$). The execution time is below 0.1 ms in all the considered cases.

The same setup is used to compare the training time of C-GMR, SEDS, and SEDSII. Obtained results are shown in Figure 3.7, where a logarithmic scale is used for the ordinate axis in order to improve the visualization. SEDS spends about 6.5 s (median value) for each motion to learn a stable DS form demonstrations. SEDSII needs 0.76 s with CLF0 and 2.7 s with CLF3. C-GMR outperforms other approaches in terms of training time with a median time of 0.58 s. Note that the training time for C-GMR in Figure 3.7 also includes the time spent to execute Algorithm 1 and to compute the demonstration area. Indeed, as shown in the previous experiment, the extra computation time introduced by C-GMR is at least three orders of magnitude smaller than the GMM time.

Generalization to different initial/target positions

One of the benefits of the motion generation based on DS is the possibility to generalize the motion to different initial and target positions. Indeed, stable DS are mathematically proven to be able

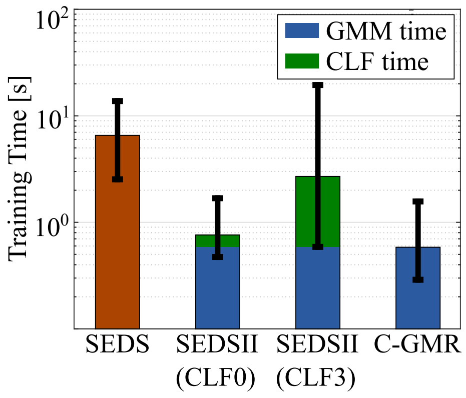


Figure 3.7.: Median training time for SEDS, SEDII, and C-GMR on the LASA dataset. Black error bars are 10% and 90% quantiles of the median value. GMM time indicates the time spent by the E–M algorithm to find the GMM parameters.

to generate convergent paths for any initial/target position. Even if the stability of C-GMR is proven in Theorem 2, it is interesting to qualitatively evaluate the generalization capabilities of C-GMR when the initial or target position changes. The generalization capabilities of C-GMR are qualitatively shown in Figure 3.8 for the S-shape motion in the LASA dataset. As shown in Figure 3.8(a), the trajectories generated by C-GMR from different initial positions always reach the desired target. This is an expected result, given the results in Theorem 2. In Figure 3.8(a), the black solid lines represent trajectories that start or enter in the demonstration ares $\mathcal{D}_r$. Due to the activation function in (3.14), the controlled DS is capable to smoothly transit between the different regions of the state space. Black dashed lines indicate trajectories that start and remain in the contracting region $\mathcal{C}_r \cup \mathcal{B}_r$. These trajectories follow exponentially converging dynamics. Figure 3.8(b) shows the generalization to the new goal position $\boldsymbol{x}^* = [-25, 25]^{\mathrm{T}}$.

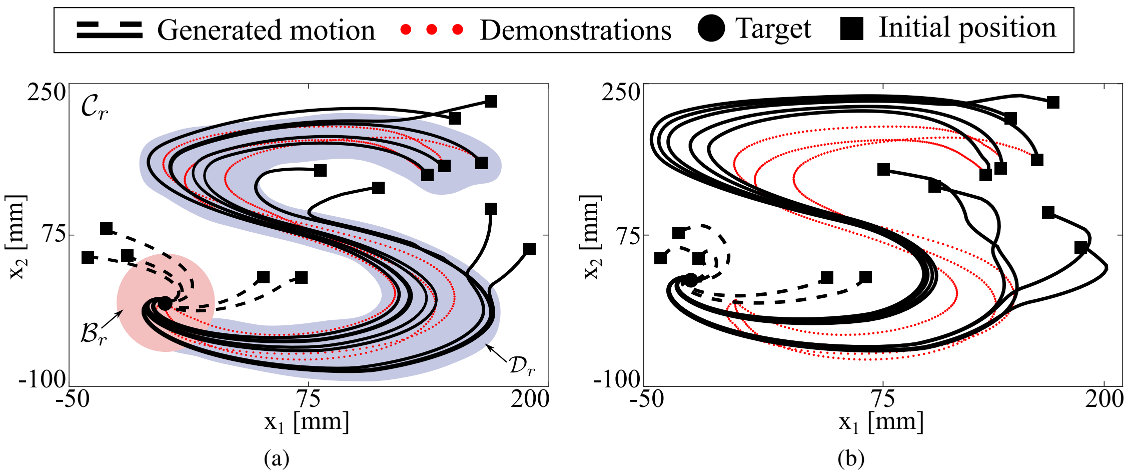
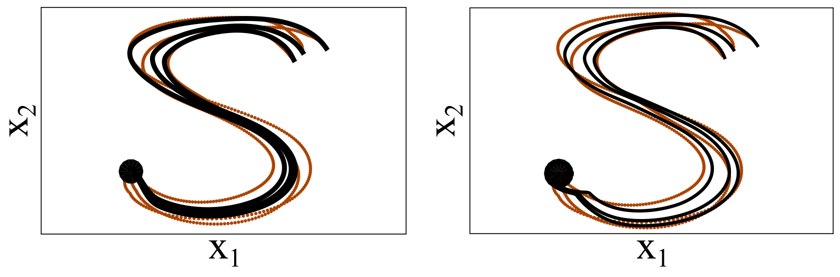


Figure 3.8.: C-GMR used to reproduce the S-shape motion from different stating (a) and target (b) positions.



(a) SEDSII CLF3 (b) C-GMR

Figure 3.9.: The first motion in MultiModel 3 is accurately reproduced with C-GMR and SEDSII CLF3. Black lines are the retrieved trajectories converging to the unique equilibrium (black bullet). Brown dots are the demonstrations.

Incremental learning

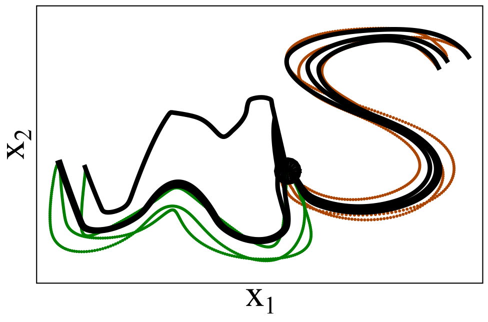
The training time results obtained in the previous experiments suggest that C-GMR can be effectively exploited for incremental learning, in a scenario where novel task demonstrations are continuously provided. The LASA dataset contains also four multi-model motions, where demonstrations show different motions in different regions of the space. As a proof of concepts, the MultiModel 3 motion from the LASA dataset is considered (see Figure 3.10). As shown in the previous experiment, SEDS is significantly slower and less accurate than the other approaches. Moreover, SEDSII CLF0 achieves poor accuracy in this case, because the complex multi-model motion often violates the constraints imposed by the Lyapunov function $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}^{0}\boldsymbol{x}$. For these reasons, only the performance of C-GMR and SEDSII CLF3 are compared in this experiment.

MultiModel 3 consists of six demonstrations that describe two motions. Two tests are performed to evaluate the generalization capabilities of SEDSII CLF3 and C-GMR. In the first test, the original GMR based DS is learned considering the first three demonstrations. As qualitatively shown in Figure 3.9, both C-GMR and SEDSII CLF3 stabilize the GMR based DS while producing trajectories close to the demonstrations. In the second test, three more demonstrations are added and the GMR based DS is re-trained[1]. Results in Figure 3.10(a) and Figure 3.11(a) show that, without re-training the CLF, SEDSII CLF3 accurately represents only one motion (reproduction error of $0.32\,\mathrm{m}^2$). As shown in Figure 3.10(b) and Figure 3.11(a), by re-training the CLF considering all the six demonstrations SEDSII CLF3 accurately represents both motions (reproduction error of $0.06\,\mathrm{m}^2$), but it is almost 6.5 times slower than SEDSII with initial CLF3 (see Figure 3.11(b)). Results for C-GMR (with recomputed control gains) are shown in Figure 3.10(c) and Figure 3.11. C-GMR has the same training time of SEDSII with initial CLF3, but it is almost 5 times more accurate ($0.064\,\mathrm{m}^2$ and $0.32\,\mathrm{m}^2$ respectively). The time to recompute control parameters, in fact, is negligible compared to the time needed to re-train the GMR. C-GMR and SEDSII with re-trained CLF achieve similar accuracy ($0.064\,\mathrm{m}^2$ and $0.06\,\mathrm{m}^2$ respectively), but the training time of SEDSII is about 6.5 times longer (2.17 s and 14.13 s respectively). Hence, C-GMR offers a good trade-off between accuracy and training time.

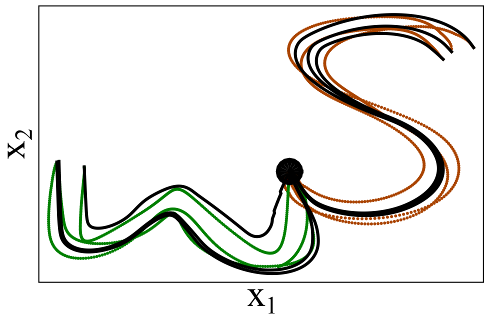
Discussion

In the conducted evaluation, C-GMR resulted to be less accurate than SEDSII. The reason is that SEDSII is optimized along the whole trajectory, while C-GMR loses accuracy when the motion

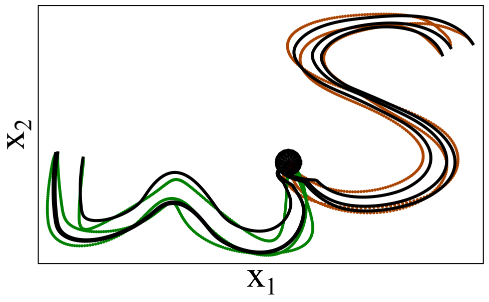
[1]Computationally efficient algorithms can be used to incrementally adjust GMM parameters when a new demonstration is provided [32, 47].



(a) SEDSII (initial CLF3)

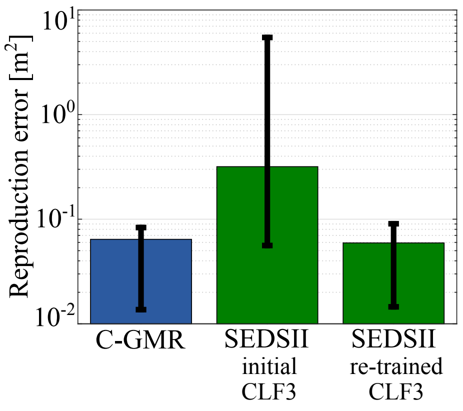


(b) SEDSII (re-trained CLF3)

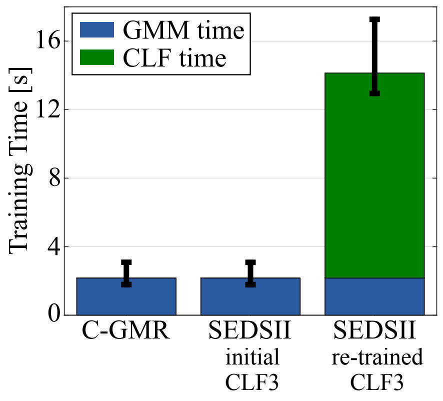


(c) C-GMR (recomputed gains)

Figure 3.10.: MultiModel 3 is accurately reproduced with C-GMR (recomputed control gains) and SEDSII (re-trained CLF3). SEDSII (initial CLF3) accurately reproduce only the trained motion (brown dots). Black lines are the retrieved trajectories converging to the unique equilibrium (black bullet). Brown dots are the first set of demonstrations, green dots the second one.



(a)



(b)

Figure 3.11.: Accuracy (a) and training time (b) for SEDII and C-GMR in the incremental learning test. The black error bars represent 10% and 90% quantiles of the median value.

enters in the ball $\mathcal{B}_r$ (see Figure 3.3) around the target. Indeed, the controlled DS converges exponentially inside $\mathcal{B}_r$. As discussed in [95], Contraction theory can be extended to consider a generic coordinate transformation $\boldsymbol{\Theta}(\boldsymbol{x})$. A possible research direction consists in finding a suitable coordinate transformation $\boldsymbol{\Theta}(\boldsymbol{x})$ that transforms the input space into a novel space where demonstrations are accurately reproduced.

The difference between C-GMR and the considered Lyapunov based approaches is that C-GMR stabilizes the dynamical system at run time, without extra learning steps. C-GMR automatically computes the control input given GMR parameters. The time required to compute the control input is, in practice, negligible. As a results, C-GMR outperforms the considered Lyapunov based approaches in terms of training time. Considering the reduced training time, a possible application of C-GMR is incremental learning, where GMR parameters have to be re-trained every time a novel skill demonstration is given. On the other hand, SEDSII is well-suited for applications where high accuracy is required, and for which the training time is not a limitation. SEDS has the worst average performance on the LASA dataset. It must be noticed that for some motions, as the JShape (the fifth motion in Figure 3.4), SEDS outperforms both SEDSII and C-GMR in terms of accuracy. In general, SEDS outperforms other approaches when the motion can be effectively represented by a quadratic Lyapunov function.

In a properly demonstrated point-to-point motion the velocity drops to zero only at the target position. In other words, the demonstrations do not exhibit spurious equilibrium points. This is the general assumption behind the motion generation with autonomous DS. An autonomous DS, in fact, is not capable to generate a motion that stops in a certain position for some seconds and then reaches another target. In case the robot has to stop for a certain time and then to reach the goal, a possible solution consists in using multiple autonomous DS (one for each attractor) and to switch among them when the robot reaches one of the equilibrium points.

3.2. Invariant representations of Cartesian trajectories

Cartesian trajectory descriptors, as the dynamical systems based representations presented in the previous section, describe the spatial motion of one or multiple points attached to the human (robot) body. As underlined in [155], Cartesian descriptors depend on *how* and *where* the same motion is executed. For instance, the same motion can be executed at different speeds, or with different amplitudes. These motion variations depend on *how* the motion is executed. Moreover, the same motion can be executed from a different starting point, or observed from a different view. These motion variations depend on *where* the motion is executed. Described motion variations reduce the effectiveness of Cartesian representations in motion recognition and reproduction problems. In order to overcome these limitations, this work proposes to map Cartesian trajectories to a space that is invariant to the aforementioned motion variations. Two invariant representations are described in the rest of this section. A theoretical comparison between different invariant representations is presented in Appendix B.

3.2.1. Unidirectional invariant representation of rigid body motion trajectories

A unidirectional invariant representation allows to transform a Cartesian trajectory into a motion descriptor invariant, for example, to roto–translations and scaling factors. However, it does not allow to reconstruct the original Cartesian trajectory given its invariant descriptor. For this reason, a unidirectional representation can be exploited for motion recognition, but not to reproduce the motion on a real robot. The invariant descriptor described in this section leverages the concept of curvature to transform a 6D Cartesian trajectory into a 2D invariant one. The resulting invariants are not affected by roto–translations and linear scale. In gesture recognition problems, the invariance to roto–translation helps to recognize gestures observed from different views, while the invariance to linear scaling factors helps to cope with the different body lengths of different users.

Consider a 6D trajectory of a rigid body where $\boldsymbol{v}_t = [v_{x,t}, v_{y,t}, v_{z,t}]^{\mathrm{T}}$ is the linear and $\boldsymbol{\omega}_t =$

$[\omega_{x,t}, \omega_{y,t}, \omega_{z,t}]^{\mathrm{T}}$ the angular velocity of the rigid body at each time instant $t = 1,\dots,T$. Given $\boldsymbol{v}_t$, an invariant representation of the rigid body translation (linear velocity) is defined as

$$\gamma_t = \frac{\|\boldsymbol{v}_t \times \dot{\boldsymbol{v}}_t\|}{\|\boldsymbol{v}_t\|^2} = \kappa(t)\,\|\boldsymbol{v}_t\|, \tag{3.16}$$

where $\boldsymbol{v}_t$ represents the time derivative of $\boldsymbol{v}_t$, and κ indicates the *curvature*. It is possible to show that γ_t in (3.16) is invariant under a constant rotation, translation, and linear scale. In other words, γ_t is invariant to affine transformations. To prove this invariance properties, it is sufficient to apply a constant affine transformation to the position $\boldsymbol{p}_t$ of the rigid body, obtaining the transformed position $\boldsymbol{p}'_t = \alpha\boldsymbol{R}\boldsymbol{p}_t + \boldsymbol{t}$, where $\alpha \neq 0$ is a scaling factor, $\boldsymbol{R}$ is a rotation matrix, and $\boldsymbol{t} = [t_x, t_y, t_z]^{\mathrm{T}}$ is a translation. The transformed velocity is then $\boldsymbol{v}'_t = \dot{\boldsymbol{p}}'_t = \alpha\boldsymbol{R}\dot{\boldsymbol{p}}_t = \alpha\boldsymbol{R}\boldsymbol{v}_t$, while the transformed acceleration is $\dot{\boldsymbol{v}}'_t = \alpha\boldsymbol{R}\dot{\boldsymbol{v}}_t$. Using the properties $(\boldsymbol{R}\boldsymbol{a}) \times (\boldsymbol{R}\boldsymbol{b}) = \boldsymbol{R}(\boldsymbol{a} \times \boldsymbol{b})$, $\|\boldsymbol{R}\boldsymbol{a}\| = \|\boldsymbol{a}\|$, and $\|\boldsymbol{v} \times \boldsymbol{w}\| = \|\boldsymbol{v}\|\,\|\boldsymbol{w}\|\,sin(\theta)$, where the vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$, it is possible to derive that

$$\gamma'_t = \frac{\|\boldsymbol{v}'_t \times \dot{\boldsymbol{v}}'_t\|}{\|\boldsymbol{v}'_t\|^2} = \frac{\left\|\alpha^2\boldsymbol{R}(\boldsymbol{v}_t \times \dot{\boldsymbol{v}}_t)\right\|}{\|\alpha\boldsymbol{R}\boldsymbol{v}_t\|^2} = \gamma_t.$$

Similarly, given $\boldsymbol{\omega}_t = [\omega_{x,t}, \omega_{y,t}, \omega_{z,t}]^{\mathrm{T}}$, an invariant representation of the rotation is defined as

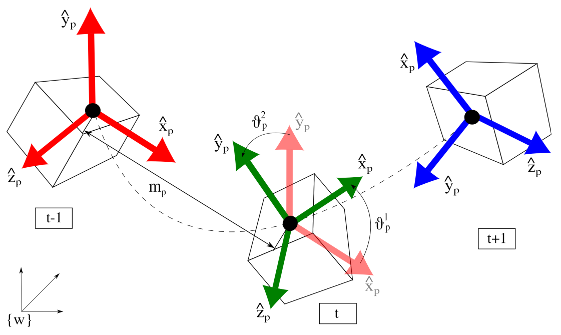
$$\xi_t = \frac{\|\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t\|}{\|\boldsymbol{\omega}_t\|^2}. \tag{3.17}$$

The representation in (3.17) is invariant to roto–translations. This result can be obtained by considering that the translation does not affect the angular velocity, and that the rotation does not affect the norm of a vector and the angle between two vectors. However, the invariance to the angular scale is not proved. This is is because the time derivative of the orientation alone does not represent the angular velocity. For example, the mapping between the derivatives of the Euler angles $\dot{\boldsymbol{e}}_t = [\dot{\phi}_t, \dot{\theta}_t, \dot{\psi}_t]^{\mathrm{T}}$ and the angular velocity $\boldsymbol{\omega}_t$ is $\boldsymbol{\omega}_t = \boldsymbol{T}(\phi_t, \theta_t, \psi_t)\dot{\boldsymbol{e}}_t$, where the matrix $\boldsymbol{T}(\cdot)$ is a non-linear mapping and it depends on the used Euler angles (see [142]).

Finally, one can notice that the values in (3.16) and (3.17) are not speed invariant. This is easily shown by considering the original velocity $\boldsymbol{v}_t$, and its scaled version $\boldsymbol{v}^s_{t_s} = \alpha\boldsymbol{v}_t$. Since the speed changes, there is a corresponding change in motion time, i.e. $t = \alpha t_s$. Hence, it holds that $\boldsymbol{v}^s_{t_s} = \alpha\boldsymbol{v}_{\alpha t_s}$ and that $\dot{\boldsymbol{v}}^s_{t_s} = \alpha^2\dot{\boldsymbol{v}}_{\alpha t_s}$. By substituting $\boldsymbol{v}^s_{t_s}$ and $\dot{\boldsymbol{v}}^s_{t_s}$ in (3.16), it is easy to verify that $\gamma^s(t_s) = \alpha\gamma_t$. Following similar steps, it is possible to show that (3.17) is not speed invariant.

3.2.2. Bidirectional invariant representation of rigid body motion trajectories

Similarly to unidirectional descriptors, bidirectional invariant representations allow to transform a Cartesian trajectory into a space where the motion exhibits some invariance properties. Moreover, a bidirectional descriptor can be converted back to the Cartesian space in order to produce a generalized version of a certain motion (see Section 3.2.4). This section presents a numerically robust trajectory descriptor called DHB (Denavit–Hartenberg inspired Bidirectional) representation. DHB is a minimal (six values) and numerically robust representation of rigid body trajectories, and it is invariant to roto–translations, linear, angular, and time scaling, and viewpoints. The key idea behind the DHB representation is the separation of the linear and angular motions of a rigid body into two distinct parts. The separation is realized by attaching two frames to the rigid body, as shown in Figure 3.12. One frame represents the position or the linear velocity in each time instant t, and it is called the *linear* frame. The other frame represents the orientation or the angular


(a) Linear frame

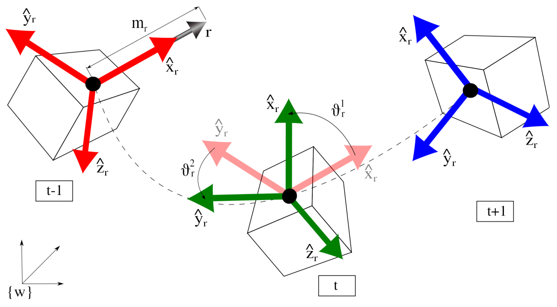

(b) Angular frame

Figure 3.12.: The linear (a) and angular (b) frames shown in three consecutive time instants.

velocity in each t, and it is called the *angular* frame. Notice that, in the rest of the section, the subscript p refers to the position and r refers to the orientation represented as a rotation vector (see Appendix A.2.1). Similarly, the subscript v indicates to the linear velocity, while ω indicates to the angular velocity. Given the linear and angular frames in each time, a six invariant values (minimal set) are computed. The invariant m_p (m_v) is the norm of the position (linear velocity) between two the consecutive time instants t and $t+1$. Similarly, m_r (m_ω) is the norm of the orientation vector (angular velocity) between t and $t+1$. The remaining four invariants, namely θ_p^i (θ_v^i) and θ_r^i (θ_ω^i), $i = 1,2$, represent the rotation of the linear (angular) frame in the consecutive time instants t and $t+1$.

Position-based Invariants

The first step to compute DHB invariants at position level is to define the linear and angular frames shown in Figure 3.12. For each time step t, the linear frame is computed by applying the following procedure:

- The x axis of the frame is the unit vector representing the normalized difference between the position vectors in t and $t+1$, i.e.

$$\hat{\boldsymbol{x}}_{p,t} = \frac{\boldsymbol{p}_{t+1} - \boldsymbol{p}_t}{\|\boldsymbol{p}_{t+1} - \boldsymbol{p}_t\|} = \frac{\Delta \boldsymbol{p}_t}{\|\Delta \boldsymbol{p}_t\|}. \tag{3.18}$$

  Note that, by construction, $\Delta \boldsymbol{p}_t$ is the linear velocity of the rigid body in a unitary time.

- The y axis represents the common normal between the x axis at the current instant t and the x axis at the next instant $t+1$. It is computed as

$$\hat{\boldsymbol{y}}_{p,t} = \frac{\hat{\boldsymbol{x}}_{p,t} \times \hat{\boldsymbol{x}}_{p,t+1}}{\|\hat{\boldsymbol{x}}_{p,t} \times \hat{\boldsymbol{x}}_{p,t+1}\|}. \tag{3.19}$$

- The z axis is orthogonal to the x and the y axes, and it is computed as

$$\hat{\boldsymbol{z}}_{p,t} = \hat{\boldsymbol{x}}_{p,t} \times \hat{\boldsymbol{y}}_{p,t}. \tag{3.20}$$

The angular frame in Figure 3.12(b) is created by applying similar steps. In particular, the x axis of the angular frame is

$$\hat{\boldsymbol{x}}_{r,t} = \frac{\Delta \boldsymbol{r}_t}{\|\Delta \boldsymbol{r}_t\|}, \tag{3.21}$$

where the rotation vector $\Delta \boldsymbol{r}_t$ is the relative orientation of the rigid body in two consecutive time instants t and $t+1$. In other words, $\Delta \boldsymbol{r}_t$ represents the angular velocity needed to rotate the body from t to $t+1$ in a unitary time. The y axis and the z axis are then computed as

$$\begin{aligned} \hat{\boldsymbol{y}}_{r,t} &= \frac{\hat{\boldsymbol{x}}_{r,t} \times \hat{\boldsymbol{x}}_{r,t+1}}{\|\hat{\boldsymbol{x}}_{r,t} \times \hat{\boldsymbol{x}}_{r,t+1}\|}, & (3.22) \\ \hat{\boldsymbol{z}}_{r,t} &= \hat{\boldsymbol{x}}_{r,t} \times \hat{\boldsymbol{y}}_{r,t}. & (3.23) \end{aligned}$$

In order to avoid discontinuities, i.e. jumps of $\pm\pi$ in subsequent time instants, the direction of the axes of the linear (angular) frames has to be modified accordingly. Given two consecutive axes $\hat{\boldsymbol{a}}_t$ and $\hat{\boldsymbol{a}}_{t+1}$, one has to set $\hat{\boldsymbol{a}}_{t+1} = -\hat{\boldsymbol{a}}_{t+1}$ if the scalar product $s = \hat{\boldsymbol{a}}_t \cdot \hat{\boldsymbol{a}}_{t+1} < 0$.

Having computed the linear and angular frames, it is possible to calculate the invariant values. The six invariants are inspired by the Denavit–Hartenberg (DH) notation [44]. One difference between DHB and DH is that, instead of applying the DH notation to the different links of a kinematic chain, DHB applies the DH notation to consecutive instants of a motion trajectory. Another difference is that the DH notation adopts one frame, while two frames are exploited by DHB to separate position and orientation. The first two invariant values are the norm of the relative positions and orientations between subsequent frames, and they are computed as

$$\begin{aligned} m_{p,t} &= \|\Delta \boldsymbol{p}_t\| = \Delta \boldsymbol{p}_t \cdot \hat{\boldsymbol{x}}_{p,t}, & (3.24) \\ m_{r,t} &= \|\Delta \boldsymbol{r}_t\| = \Delta \boldsymbol{r}_t \cdot \hat{\boldsymbol{x}}_{r,t}, & (3.25) \end{aligned}$$

where $\hat{\boldsymbol{x}}_{p,t}$ and $\hat{\boldsymbol{x}}_{r,t}$ are the axes in (3.18) and (3.21) respectively. The invariants m_p and m_r in (3.24) and (3.25) describe the translation of the linear and angular frames respectively. The rotation of the linear and angular frames is described by four more values.

For the linear frame, the axis $\hat{\boldsymbol{y}}_{p,t}$ lies on the common normal between $\hat{\boldsymbol{x}}_{p,t}$ and $\hat{\boldsymbol{x}}_{p,t+1}$. According to the DH notation, only the rotations about the x and y axes are needed to align the frames at t and $t+1$. As shown in Figure 3.12(a), to align $\hat{\boldsymbol{x}}_{p,t}$ to $\hat{\boldsymbol{x}}_{p,t+1}$ the linear frame has to rotate an angle θ_p^1 about $\hat{\boldsymbol{y}}_{p,t}$. The signed value of θ_p^1 is given by

$$\begin{aligned} \theta_{p,t}^1 &= \arctan\left(\frac{\hat{\boldsymbol{x}}_{p,t} \times \hat{\boldsymbol{x}}_{p,t+1}}{\hat{\boldsymbol{x}}_{p,t} \cdot \hat{\boldsymbol{x}}_{p,t+1}} \cdot \hat{\boldsymbol{y}}_{p,t}\right) \\ &= \arctan\left(\frac{\|\hat{\boldsymbol{x}}_{p,t}\|\|\hat{\boldsymbol{x}}_{p,t+1}\|\sin(\theta_{p,t}^1)\hat{\boldsymbol{y}}_{p,t}}{\|\hat{\boldsymbol{x}}_{p,t}\|\|\hat{\boldsymbol{x}}_{p,t+1}\|\cos(\theta_{p,t}^1)} \cdot \hat{\boldsymbol{y}}_{p,t}\right) \qquad (3.26) \\ &= \arctan\left(\frac{\sin(\theta_{p,t}^1)}{\cos(\theta_{p,t}^1)}\right), \end{aligned}$$

where the relationships $\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\cos(\theta_{ab})$, $\boldsymbol{a} \times \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\sin(\theta_{ab})\hat{\boldsymbol{n}}$, and (3.19) have been used. A further rotation θ_p^2 about $\hat{\boldsymbol{x}}_{p,t+1}$ is required to align $\hat{\boldsymbol{y}}_{p,t}$ to $\hat{\boldsymbol{y}}_{p,t+1}$. The signed value of θ_p^2 is computed as

$$\theta_{p,t}^2 = \arctan\left(\frac{\hat{\boldsymbol{y}}_{p,t} \times \hat{\boldsymbol{y}}_{p,t+1}}{\hat{\boldsymbol{y}}_{p,t} \cdot \hat{\boldsymbol{y}}_{p,t+1}} \cdot \hat{\boldsymbol{x}}_{p,t+1}\right). \qquad (3.27)$$

A similar reasoning is applied to to align the angular frames between t and $t+1$. The resulting invariants are

$$\begin{aligned} \theta_{r,t}^1 &= \arctan\left(\frac{\hat{\boldsymbol{x}}_{r,t} \times \hat{\boldsymbol{x}}_{r,t+1}}{\hat{\boldsymbol{x}}_{r,t} \cdot \hat{\boldsymbol{x}}_{r,t+1}} \cdot \hat{\boldsymbol{y}}_{r,t}\right), & (3.28) \\ \theta_{r,t}^2 &= \arctan\left(\frac{\hat{\boldsymbol{y}}_{r,t} \times \hat{\boldsymbol{y}}_{r,t+1}}{\hat{\boldsymbol{y}}_{r,t} \cdot \hat{\boldsymbol{y}}_{r,t+1}} \cdot \hat{\boldsymbol{x}}_{r,t+1}\right). & (3.29) \end{aligned}$$

Note that the definitions of the invariants in (3.26) and (3.28), as well as those in (3.27) and (3.29), are formally the same.

Velocity-based Invariants

In some cases, it is possible that the linear $\boldsymbol{v}_t$ and angular $\boldsymbol{\omega}_t$ velocities are directly available from the sensors. In this case, the DHB invariants can be directly computed from the velocity data, without any extra integration step. Being computed starting from the velocity, these invariant values are referred as velocity-based invariants. The approach used to calculate the position-based invariants can be easily adapted to the velocity-based invariants. The main differences between the two approaches are discussed as follows. Similarly to position-based invariants two frames are attached to the rigid body. However, for velocity-based invariants, the frames are used to separate linear and angular velocity vectors. The axes of the linear frame are computed as

$$\hat{\boldsymbol{x}}_{v,t} = \frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|}, \tag{3.30}$$

$$\hat{\boldsymbol{y}}_{v,t} = \frac{\hat{\boldsymbol{x}}_{v,t} \times \hat{\boldsymbol{x}}_{v,t+1}}{\|\hat{\boldsymbol{x}}_{v,t} \times \hat{\boldsymbol{x}}_{v,t+1}\|}, \tag{3.31}$$

$$\hat{\boldsymbol{z}}_{v,t} = \hat{\boldsymbol{x}}_{v,t} \times \hat{\boldsymbol{y}}_{v,t}. \tag{3.32}$$

The axes of the angular frame are given by

$$\hat{\boldsymbol{x}}_{\omega,t} = \frac{\boldsymbol{\omega}_t}{\|\boldsymbol{\omega}_t\|}, \tag{3.33}$$

$$\hat{\boldsymbol{y}}_{\omega,t} = \frac{\hat{\boldsymbol{x}}_{\omega,t} \times \hat{\boldsymbol{x}}_{\omega,t+1}}{\|\hat{\boldsymbol{x}}_{\omega,t} \times \hat{\boldsymbol{x}}_{\omega,t+1}\|}, \tag{3.34}$$

$$\hat{\boldsymbol{z}}_{\omega,t} = \hat{\boldsymbol{x}}_{\omega,t} \times \hat{\boldsymbol{y}}_{\omega,t}. \tag{3.35}$$

Note the strong similarities between (3.18)-(3.20) and (3.30)-(3.32), as well as between (3.21)-(3.23) and (3.33)-(3.35).

Given the two frames, the six velocity-based invariants are computed. Two values are the norm of the linear and angular velocities, and they are computed as

$$m_{v,t} = \|\boldsymbol{v}_t\| = \boldsymbol{v}_t \cdot \hat{\boldsymbol{x}}_{v,t}, \tag{3.36}$$

$$m_{\omega,t} = \|\boldsymbol{\omega}_t\| = \boldsymbol{\omega}_t \cdot \hat{\boldsymbol{x}}_{\omega,t}. \tag{3.37}$$

As for the position-based invariants, the remaining four values are used to align the linear and angular frames in two consecutive time instants. The invariants for the linear velocity are

$$\theta^1_{v,t} = \arctan\left(\frac{\hat{\boldsymbol{x}}_{v,t} \times \hat{\boldsymbol{x}}_{v,t+1}}{\hat{\boldsymbol{x}}_{v,t} \cdot \hat{\boldsymbol{x}}_{v,t+1}} \cdot \hat{\boldsymbol{y}}_{v,t}\right), \tag{3.38}$$

$$\theta^2_{v,t} = \arctan\left(\frac{\hat{\boldsymbol{y}}_{v,t} \times \hat{\boldsymbol{y}}_{v,t+1}}{\hat{\boldsymbol{y}}_{v,t} \cdot \hat{\boldsymbol{y}}_{v,t+1}} \cdot \hat{\boldsymbol{x}}_{v,t+1}\right), \tag{3.39}$$

while the invariants for the angular velocity are

$$\theta^1_{\omega,t} = \arctan\left(\frac{\hat{\boldsymbol{x}}_{\omega,t} \times \hat{\boldsymbol{x}}_{\omega,t+1}}{\hat{\boldsymbol{x}}_{\omega,t} \cdot \hat{\boldsymbol{x}}_{\omega,t+1}} \cdot \hat{\boldsymbol{y}}_{\omega,t}\right), \tag{3.40}$$

$$\theta^2_{v,t} = \arctan\left(\frac{\hat{\boldsymbol{y}}_{\omega,t} \times \hat{\boldsymbol{y}}_{\omega,t+1}}{\hat{\boldsymbol{y}}_{\omega,t} \cdot \hat{\boldsymbol{y}}_{\omega,t+1}} \cdot \hat{\boldsymbol{x}}_{\omega,t+1}\right). \tag{3.41}$$

It is worth noticing that $\theta^1_{v,t}$ and $\theta^2_{v,t}$ are computed by substituting $\hat{\boldsymbol{x}}_{p,t}$ with $\hat{\boldsymbol{x}}_{v,t}$ and $\hat{\boldsymbol{y}}_{p,t}$ with $\hat{\boldsymbol{y}}_{v,t}$ in (3.26) and (3.27). Similarly, $\theta^1_{\omega,t}$ and $\theta^2_{\omega,t}$ are computed by substituting $\hat{\boldsymbol{x}}_{r,t}$ with $\hat{\boldsymbol{x}}_{\omega,t}$ and $\hat{\boldsymbol{y}}_{r,t}$ with $\hat{\boldsymbol{y}}_{\omega,t}$ in (3.28) and (3.29).

In the discrete time domain, a simple relationship exists between position- and velocity-based invariants. To find the existing relation, one has to consider that $\Delta \boldsymbol{p}$ and $\Delta \boldsymbol{r}$ represent the linear and angular velocities between two consecutive frames in a unitary time. Given this, from (3.24) and (3.25), it is easy to derive that

$$m_{v,t} = \|\boldsymbol{v}_t\| = \frac{\|\Delta \boldsymbol{p}_t\|}{\Delta t} = \frac{m_{p,t}}{\Delta t}, \tag{3.42}$$

$$m_{\omega,t} = \|\boldsymbol{\omega}_t\| = \frac{\|\Delta \boldsymbol{r}_t\|}{\Delta t} = \frac{m_{r,t}}{\Delta t}, \tag{3.43}$$

where Δt is the sampling time. Moreover, the remaining four invariants are the same in both the representations. This is shown by considering that $\hat{\boldsymbol{x}}_{v,t} = \hat{\boldsymbol{x}}_{p,t}$ (see (3.18) and (3.30)), while from (3.21) and (3.33) it comes that $\hat{\boldsymbol{x}}_{\omega,t} = \hat{\boldsymbol{x}}_{r,t}$. Finally, comparing (3.38)-(3.41) with (3.26)-(3.29), and considering that the y axis definitions are formally the same for all the invariants, it follows that $[\theta_v^1,\ \theta_v^2,\ \theta_\omega^1,\ \theta_\omega^2] = [\theta_p^1,\ \theta_p^2,\ \theta_r^1,\ \theta_r^2]$.

DHB Representation in Closed Form

DHB invariants were presented in a way that is useful to understand the physical meaning of each invariant value. However, a closed form of the DHB representation is useful to simplify the formulation and an eventual implementation. In order to provide a closed form or, in other words, a set of equations mapping the Cartesian space (position/velocity) into the invariant space, one can rewrite the expression of $\hat{\boldsymbol{y}}_{v,t}$ in (3.31) as

$$\hat{\boldsymbol{y}}_{v,t} = \frac{\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}}{\|\boldsymbol{v}_t\|\|\boldsymbol{v}_{t+1}\|} \frac{\|\boldsymbol{v}_t\|\|\boldsymbol{v}_{t+1}\|}{\|\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}\|} = \frac{\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}}{\|\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}\|}, \tag{3.44}$$

where the property $\hat{\boldsymbol{x}}_{v,t} = \frac{\boldsymbol{v}_t}{\|\boldsymbol{v}_t\|}$ is used. Considering (3.38) and (3.44), $\theta_{v,t}^1$ becomes

$$\begin{aligned}
\theta_{v,t}^1 &= \arctan\left(\frac{\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}}{\|\boldsymbol{v}_t\|\|\boldsymbol{v}_{t+1}\|} \frac{\|\boldsymbol{v}_t\|\|\boldsymbol{v}_{t+1}\|}{\boldsymbol{v}_t \cdot \boldsymbol{v}_{t+1}} \cdot \hat{\boldsymbol{y}}_{v,t} \right) \\
&= \arctan\left(\frac{\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}}{\boldsymbol{v}_t \cdot \boldsymbol{v}_{t+1}} \cdot \frac{\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}}{\|\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}\|} \right) \\
&= \arctan\left(\frac{\|\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}\|}{\boldsymbol{v}_t \cdot \boldsymbol{v}_{t+1}} \right).
\end{aligned} \tag{3.45}$$

Following similar steps, one can show that closed forms for $\theta_{p,t}^1$, $\theta_{r,t}^1$, $\theta_{\omega,t}^1$ are formally the same as (3.45).

The expression of $\theta_{v,t}^2$ in a closed form is slightly more complicated. Hence, it is useful to first focus on the division in (3.39). Considering (3.44), it is straightforward to derive that

$$\begin{aligned}
\frac{\hat{\boldsymbol{y}}_{v,t} \times \hat{\boldsymbol{y}}_{v,t+1}}{\hat{\boldsymbol{y}}_{v,t} \cdot \hat{\boldsymbol{y}}_{v,t+1}} &= \frac{(\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}) \times (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})}{(\boldsymbol{v}_t \times \boldsymbol{v}_{t+1}) \cdot (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})} \\
&= \frac{-(\boldsymbol{v}_{t+1} \times \boldsymbol{v}_t) \times (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})}{-(\boldsymbol{v}_{t+1} \times \boldsymbol{v}_t) \cdot (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})} \\
&= \frac{[\boldsymbol{v}_{t+1} \cdot (\boldsymbol{v}_t \times \boldsymbol{v}_{t+2})]\, \boldsymbol{v}_{t+1}}{(\boldsymbol{v}_{t+1} \times \boldsymbol{v}_t) \cdot (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})},
\end{aligned} \tag{3.46}$$

where the relationships $\boldsymbol{a} \times \boldsymbol{b} = -\boldsymbol{b} \times \boldsymbol{a}$ and $(\boldsymbol{a} \times \boldsymbol{b}) \times (\boldsymbol{a} \times \boldsymbol{c}) = \boldsymbol{a} \cdot (\boldsymbol{b} \times \boldsymbol{c})\boldsymbol{a}$ have been used. The closed form expression

$$\theta_{v,t}^2 = \arctan\left(\frac{\|\boldsymbol{v}_{t+1}\| \boldsymbol{v}_{t+1} \cdot (\boldsymbol{v}_t \times \boldsymbol{v}_{t+2})}{(\boldsymbol{v}_{t+1} \times \boldsymbol{v}_t) \cdot (\boldsymbol{v}_{t+1} \times \boldsymbol{v}_{t+2})} \right), \tag{3.47}$$

Algorithm 2 DHB Invariant Representation

```
Position-based invariants
Input: a set of relative positions $\{\Delta\boldsymbol{p}\}_t$ and rotation
vectors $\{\Delta\boldsymbol{r}\}_t$, $t = 1,\dots,N$
return cartesianToDHB($\{\Delta\boldsymbol{p}\}_t, \{\Delta\boldsymbol{r}\}_t$)

Velocity-based invariants
Input: a set of linear $\{\boldsymbol{v}\}_t$ and angular $\{\boldsymbol{\omega}\}_t$, $t = 1,\dots,N$
velocities
return cartesianToDHB($\{\boldsymbol{v}\}_t, \{\boldsymbol{\omega}\}_t$)

cartesianToDHB($\{\boldsymbol{l}\}_t, \{\boldsymbol{a}\}_t$)
 1. $[\{m_l\}_t, \{\theta_l^1\}_t, \{\theta_l^2\}_t]$ = computeInvariants($\{\boldsymbol{l}\}_t$)
 2. $[\{m_a\}_t, \{\theta_a^1\}_t, \{\theta_a^2\}_t]$ = computeInvariants($\{\boldsymbol{a}\}_t$)
return $\{m_l\}_t, \{\theta_l^1\}_t, \{\theta_l^2\}_t, \{m_a\}_t, \{\theta_a^1\}_t, \{\theta_a^2\}_t$

computeInvariants($\{\boldsymbol{u}\}_t$):
 3. for all $t \in [1, N-2]$ do
 4.     $m_{u,t} = \|\boldsymbol{u}_t\|$
 5.     $\theta_{u,t}^1 = \arctan\left(\frac{\|\boldsymbol{u}_{u,t} \times \boldsymbol{u}_{u,t+1}\|}{\boldsymbol{u}_{u,t} \cdot \boldsymbol{u}_{u,t+1}}\right)$
 6.     $\theta_{u,t}^2 = \arctan\left(\frac{\|\boldsymbol{u}_{u,t+1}\| \boldsymbol{u}_{u,t+1} \cdot (\boldsymbol{u}_{u,t} \times \boldsymbol{u}_{u,t+2})}{(\boldsymbol{u}_{u,t+1} \times \boldsymbol{u}_{u,t}) \cdot (\boldsymbol{u}_{u,t+1} \times \boldsymbol{u}_{u,t+2})}\right)$
 7. end for
return $\{m_u\}_t, \{\theta_u^1\}_t, \{\theta_u^2\}_t$
```

is finally obtained by multiplying (3.46) by $\hat{\boldsymbol{x}}_{v,t+1}$ and taking the arc tangent. Following similar steps, it is easy to show that closed forms for $\theta_{p,t}^2$, $\theta_{r,t}^2$, $\theta_{\omega,t}^2$ are formally the same as (3.47).

The described procedure to calculate position- and velocity-based invariants in closed form is summarized in Algorithm 2. Note that, in Algorithm 2, $\boldsymbol{u}$ refers to a generic vector, l stands for *linear* and a stands for *angular*. The approach is analogous for both position- and velocity-based invariants. The function *computeInvariants* takes as input a set of vectors (positions, rotation vectors, linear, or angular velocities) and returns the related set of three invariants. The function *cartesianToDHB* just calls twice *computeInvariants* to compute the complete set of invariant values (six for each time instant).

Trajectory Reconstruction

As already mentioned, DHB is a bidirectional representation. Given the position-based invariants, the original pose (position and orientation) trajectory of the rigid body in a reference (world) frame is reconstructed in three steps. First, the pose of the linear and angular frames in each instant t is computed as

$$\boldsymbol{H}_{p,t} = \begin{bmatrix} \boldsymbol{R}_y(\theta_p^1)\boldsymbol{R}_x(\theta_p^2) & \boldsymbol{m}_p \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}, \tag{3.48}$$

$$\boldsymbol{H}_{r,t} = \begin{bmatrix} \boldsymbol{R}_y(\theta_r^1)\boldsymbol{R}_x(\theta_r^2) & \boldsymbol{m}_r \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix}, \tag{3.49}$$

where $\boldsymbol{m}_p = [m_p, 0, 0]^{\mathrm{T}}$, $\boldsymbol{m}_r = [m_r, 0, 0]^{\mathrm{T}}$, $\boldsymbol{R}_y(\alpha)$ and $\boldsymbol{R}_x(\alpha)$ are the elementary rotations of an angle α about the y and x axis [142], and $\boldsymbol{0}^{\mathrm{T}} = [0, 0, 0]$. Note that the matrices in (3.48) and (3.49) are formally the same except for the fourth value of the fourth column. Further details about this choice are given later in this section.

The second step requires the pose of the linear $\boldsymbol{H}_{p,1}$ and angular $\boldsymbol{H}_{r,1}$ frames with respect to the world frame in the first time instant. $\boldsymbol{H}_{p,1}$ and $\boldsymbol{H}_{r,1}$ are computed to calculate the invariant values and stored for later used. Given $\boldsymbol{H}_{p,1}$ and $\boldsymbol{H}_{r,1}$, it is possible to compute

$$\boldsymbol{H}^{w}_{p,t} = \boldsymbol{H}_{p,1} \cdot \boldsymbol{H}_{p,2} \cdot \ldots \cdot \boldsymbol{H}_{p,t}, \tag{3.50}$$

$$\boldsymbol{H}^{w}_{r,t} = \boldsymbol{H}_{r,1} \cdot \boldsymbol{H}_{r,2} \cdot \ldots \cdot \boldsymbol{H}_{r,t}, \tag{3.51}$$

where $\boldsymbol{H}^{w}_{p,t}$ and $\boldsymbol{H}^{w}_{r,t}$ are the pose of the linear and angular frames with respect to the world frame in a time instant t.

Finally, the Cartesian position in a generic time instant t is given by

$$\boldsymbol{p}_t = \boldsymbol{R}^{w}_{p,1}\Delta\boldsymbol{p}_1 + \ldots + \boldsymbol{R}^{w}_{p,t}\Delta\boldsymbol{p}_t = \boldsymbol{H}^{w}_{p,t}[1:3,4], \tag{3.52}$$

where $\boldsymbol{R}^{w}_{p,i}$ is the rotation of the linear frame with respect to the world frame at $t = i$, and $\boldsymbol{H}^{w}_{p,t}[1:3,4]$ are the first three elements of the fourth column of $\boldsymbol{H}^{w}_{p,t}$ in (3.50). The absolute orientation of the rigid body cannot be reconstructed by following the approach used to reconstruct the position. The reason is that $\Delta\boldsymbol{R}_t + \Delta\boldsymbol{R}_{t+1} \neq \boldsymbol{R}_{t+1}$. Having chosen $\boldsymbol{H}_{r,t}[4,4] = 0$ in (3.49), it is possible to extract the relative rotation vector

$$\Delta\boldsymbol{R}_t = \boldsymbol{H}^{w}_{r,t}[1:3,4] \tag{3.53}$$

from (3.51). The orientation of the rigid body with respect to the world frame in a time instant t is computed as $\boldsymbol{R}^{w}_{t} = \exp(\Delta\boldsymbol{R}_1) \cdot \ldots \cdot \exp(\Delta\boldsymbol{R}_t)$, where the exponential mapping $\exp(\boldsymbol{R})$ transforms a rotation vector into a rotation matrix (see Appendix A.2.1).

A similar procedure is applied to reconstruct the Cartesian velocity of the rigid body from velocity-based invariants. The first step consists in computing the pose of the linear and angular frames in each time instant, i.e.

$$\boldsymbol{H}_{v,t} = \begin{bmatrix} \boldsymbol{R}_y(\theta^1_v)\boldsymbol{R}_x(\theta^2_v) & \boldsymbol{m}_v \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix}, \tag{3.54}$$

$$\boldsymbol{H}_{\omega,t} = \begin{bmatrix} \boldsymbol{R}_y(\theta^1_\omega)\boldsymbol{R}_x(\theta^2_\omega) & \boldsymbol{m}_\omega \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix}, \tag{3.55}$$

where $\boldsymbol{m}_v = [m_v,\, 0,\, 0]^{\mathrm{T}}$, $\boldsymbol{m}_\omega = [m_\omega,\, 0,\, 0]^{\mathrm{T}}$. The second step requires the pose of the linear $\boldsymbol{H}_{v,1}$ and angular $\boldsymbol{H}_{\omega,1}$ frames with respect to the world frame in the first time instant. These poses are used to calculate the invariants and are stored for later use. Given $\boldsymbol{H}_{v,1}$ and $\boldsymbol{H}_{\omega,1}$, it is possible to compute

$$\boldsymbol{H}^{w}_{v,t} = \boldsymbol{H}_{v,1} \cdot \ldots \cdot \boldsymbol{H}_{v,t}, \quad \boldsymbol{H}^{w}_{\omega,t} = \boldsymbol{H}_{\omega,1} \cdot \ldots \cdot \boldsymbol{H}_{\omega,t}. \tag{3.56}$$

The velocity in a time instant t is then computed as $\boldsymbol{v}_t = \boldsymbol{H}^{w}_{v,t}[1:3,4]$ and $\boldsymbol{\omega}_t = \boldsymbol{H}^{w}_{\omega,t}[1:3,4]$.

The described approach for Cartesian motion reconstruction can be extended to generate affine transformed instances of a Cartesian motion from the same invariant values. For instance, the procedure to reconstruct the Cartesian position is modified to generate shorter or longer trajectories by multiplying the vector $\boldsymbol{m}_p$ in (3.48) by a scaling factor, obtaining the new vector $\boldsymbol{m}^{s}_{p} = [s_p m_p,\, 0,\, 0]^{\mathrm{T}}$. Moreover, roto–translated instances of the Cartesian trajectory are generated by arbitrary roto–translating the initial pose $\boldsymbol{H}_{p,1}$ in (3.50), i.e. $\boldsymbol{H}'_{p,1} = \boldsymbol{H}^{a}_{p,1}\boldsymbol{H}_{p,1}$, where $\boldsymbol{H}^{a}_{p,1}$ is an arbitrary transformation. Note that a similar approach applies for the reconstruction of the orientation, as well as for the reconstruction of linear and angular velocities. The presented extensions of the reconstruction procedure are exploited in Section 3.2.4 to generate different trajectory instances from a single invariant trajectory. The procedure is also exploited to reproduce human demonstrations on different robotics systems, overcoming the limitations of Cartesian descriptors that depend on the absolute pose of the rigid body in the world frame.

Invariance Properties

Position- and velocity-based invariants share the same invariance properties. Hence, in the description of their properties, the symbol $\boldsymbol{u}$ is used to refer to a generic vector (position or velocity), the subscript l to refer to the three linear invariants, and a to refer to the three angular invariants.

Reference frame and Viewpoint - Cartesian trajectories depend on the choice of the reference frame. When the motion is observed by a camera, the reference frame is usually the frame attached to the camera (viewpoint). DHB is a coordinate-free representation of rigid body motions, since DHB invariants do not depend on roto–translations of the world frame and on the viewpoint. In order to prove this property, recall that a translations of the reference frame (viewpoint) does not affect the relative position between two frames, the orientation of the rigid body, as well as its linear and angular velocities. Hence, DHB invariants are not affected by constant translations. In order to prove the invariance to rotations, an arbitrary rotation $\boldsymbol{R}$ is applied to the vector $\boldsymbol{u}$. Considering that a rotation does not affect the norm of a vector, it is possible to write

$$m_{Rl} = \|\boldsymbol{R}\boldsymbol{u}_l\| = \|\boldsymbol{R}\|\,\|\boldsymbol{u}_l\| = \|\boldsymbol{u}_l\| = m_l, \tag{3.57}$$

$$m_{Ra} = \|\boldsymbol{R}\boldsymbol{u}_a\| = \|\boldsymbol{R}\|\,\|\boldsymbol{u}_a\| = \|\boldsymbol{u}_a\| = m_a, \tag{3.58}$$

where the property $\|\boldsymbol{R}\| = 1$ is used. Note that, in this context, the vector $\boldsymbol{u}_l$ in (3.57) represents either $\Delta\mathbf{p}$ or $\mathbf{v}$, while the vector $\boldsymbol{u}_a$ in (3.58) represents either $\Delta\boldsymbol{R}$ or $\boldsymbol{\omega}$. As previously discussed, the other invariant values $\theta_n^i,\ n = l,a,\ i = 1,2$ are angles between two vectors. It is easy to show that

$$\theta_{Rl}^1 = \arctan\left(\frac{\boldsymbol{R}\hat{\boldsymbol{x}}_{l,t} \times \boldsymbol{R}\hat{\boldsymbol{x}}_{l,t+1}}{\boldsymbol{R}\hat{\boldsymbol{x}}_{l,t} \cdot \boldsymbol{R}\hat{\boldsymbol{x}}_{l,t+1}} \cdot \boldsymbol{R}\hat{\boldsymbol{x}}_{l,t}\right) = \arctan\left(\frac{(\hat{\boldsymbol{x}}_{l,t} \times \hat{\boldsymbol{x}}_{l,t+1})^T}{\hat{\boldsymbol{x}}_{l,t}^T\boldsymbol{R}^T \cdot \boldsymbol{R}\hat{\boldsymbol{x}}_{l,t+1}}\boldsymbol{R}^T \cdot \boldsymbol{R}\hat{\boldsymbol{y}}_{l,t}\right) = \theta_l^1, \tag{3.59}$$

where $\hat{\boldsymbol{x}}_{l,t}$ and $\hat{\boldsymbol{y}}_{l,t}$ are the x and y axes of the linear frame respectively. Following the reasoning in (3.59), it is easy to show the invariance of θ_l^2, θ_a^1, and θ_a^2.

Time, linear, and angular scale - The same motion can be executed at different speeds, which generates motion trajectories with different time scales. Hence, the time scale invariance is of importance to compare motions performed at different speeds. As proposed in [42, 155], a dimensionless time can be defined as

$$t' = \frac{t}{t_f}, \tag{3.60}$$

where t_f is the total time duration of the motion. DHB invariants are made independent on the time scale by multiplying each m_n^i and θ_n^i by t_f and by substituting t with t'.

The invariance to scaling factors helps to recognize motions performed by different users, i.e. subjects with different kinematic structures [135]. The four θ_n^i are angles between vectors, and they are independent on linear and angular scales. Indeed, it is well-known that the scaling of two vectors does not affect the angle between them. The two $m_n^i,\ n = l,a,\ i = 1,2$, values are made invariant to scaling factors by

$$m'_{l,t} = \frac{m_{l,t}}{\int_{t=0}^{t_f} |m_{l,t}|}, \quad m'_{a,t} = \frac{m_{a,t}}{\int_{t=0}^{t_f} |m_{a,t}|}, \tag{3.61}$$

where $\int_{t=0}^{t_f} |m_{l,t}|$ and $\int_{t=0}^{t_f} |m_{a,t}|$ are the linear and angular scale of motion respectively, and t_f is the total duration of the motion. Note that, in the discrete time case, (3.61) is replaced by

$$m'_{l,t} = \frac{m_{l,t}}{\sum_{t=0}^{t_f} |m_{l,t}|}, \quad m'_{a,t} = \frac{m_{a,t}}{\sum_{t=0}^{t_f} |m_{a,t}|}. \tag{3.62}$$

Speed invariance - The invariance to the speed of execution, or the motion profile, is obtained by expressing the invariant values as a function of a degree of advancement (see, for example, [155]). Speed invariance can be achieved only in theory, because in practice the discrete sampling time of real sensors strongly affects motions executed at different speeds and the resulting invariants. This is independent on the employed invariant representation. An example is shown in Figure 3.13, where a sinusoidal trajectory (blue line) is sampled with sampling times Δt and $2\Delta t$. The green line is the reconstructed signal for Δt, while the red line is the signal obtained for $2\Delta t$. It is clear that different sampling rates generate different signals (green and red lines). Given different input signals, any representation will generate different invariant descriptors.

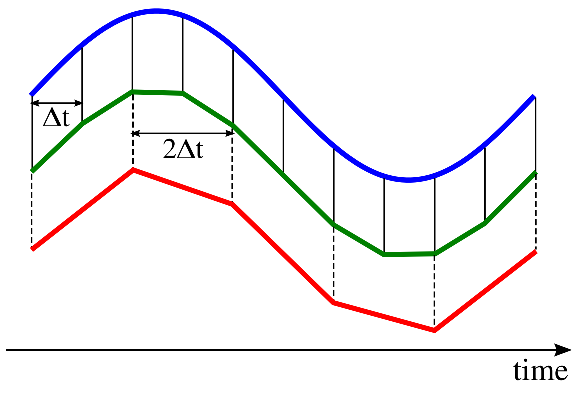


Figure 3.13.: A sinusoidal signal (blue line) is sampled with different sampling rates. The green line is the reconstructed signal obtained by sampling the blue line every Δt seconds. The red line is obtained with a sampling time of $2\Delta t$.

Reverse motion - In some cases it is useful to have the same representation for motions executed in a direction or in the reverse (opposite) one. Invariance to the direction of motion for m_l and m_a is achieved by considering them in the reverse order

$$m_{l,t}^{rev} = m_{l,(t_f-t)}, \quad m_{a,t}^{rev} = m_{a,(t_f-t)}, \tag{3.63}$$

where t_f is the total duration of the motion. The four θ_n^i invariants are also considered in the reverse order, but they have to be shifted by 1 or 2 samples, i.e.

$$\theta_{n,t}^{1,rev} = \theta_{n,(t_f-t+1)}^{1}, \quad n = l, a, \tag{3.64}$$

$$\theta_{n,t}^{2,rev} = \theta_{n,(t_f-t+2)}^{2}, \quad n = l, a. \tag{3.65}$$

Reference point - Similarly to Cartesian trajectories, $m_{l,t}$, $\theta_{l,t}^1$, and $\theta_{l,t}^2$ depend on the choice of the reference point used to describe the motion. In other words, different invariants are obtained if the same motion is referred to different points. To overcome this limitation, one has to ensure that the reference point is always the same (or slightly varying) during the entire motion. Note that the representation in [42] is invariant to the reference point, while EFS invariants also depend on the choice of the reference point.

Samples delay - DHB representation requires the Cartesian data in future (or past) time instants, introducing a samples delay. In particular, to compute the position-based invariants in a time instant t one has to know the Cartesian trajectory from t to $t+3$, introducing a delay of three samples. Indeed, the angles $\theta^2_{n,t},\ n = l,a$ in (3.27) and (3.29) depend on $\hat{\boldsymbol{y}}_{n,t+1},\ n = l,a$ and the axes $\hat{\boldsymbol{y}}_{n,t+1}$ in (3.19) and (3.22) depend on $\hat{\boldsymbol{x}}_{n,t+2},\ n = l,a$. From (3.18) and (3.21) it is clear that the computation of $\hat{\boldsymbol{x}}_{n,t+2},\ n = l,a$ requires the position and the orientation of the rigid body in $t+3$. Following a similar reasoning, recalling that the axes of the linear and angular frames have formally the same definition in both the invariant representations, and considering (3.30) and (3.33), it is easy to verify that the velocity-based invariants in t requires the velocities from t to $t+2$. Hence, a delay of two samples is introduced. It is worth noticing that, when the velocities have to be computed by numerical differentiation, position- and velocity-based invariants have the same delay of three samples. The invariants proposed in [42, 155] depend only on the current time instant. However, for each instant t, they require the third-order derivative of the position and orientation. When the time derivatives have to be computed by numerical differentiation of the position and orientation, the same delay of three samples is introduced.

Computational cost - The computational cost of the DHB representation (see Algorithm 2) is $O(N(M(n)(96+4\log(n))))$, where $M(n)$ is the computational cost of the multiplication (division) depending on the number of digits n used to represent real numbers. N is the number of samples in the Cartesian trajectory. The computational cost is computed neglecting the summation and subtraction operations in Algorithm 2. For comparison, the algorithm in [42] has a complexity of $O(N(180M(n)))$, while EFS invariants have a complexity of $O(N(100M(n)))$ (see Table B.1).

Special Motions

In some situations, one of the axes of linear or angular frames cannot be defined, and the DHB representation cannot be computed. These situations are the so-called singular cases or singularities. Considering the definition of linear and angular frames in (3.18)–(3.20) and in (3.30)–(3.32), it is clear that singularities occur when one axis cannot be normalized because the norm (denominator) drops to zero. In what follows, singular cases are analyzed and effective solutions proposed for each case. The DHB representation of the singular cases is summarized in Table 3.1.

Table 3.1.: DHB representation of special motions.

Pure translation	m_l	θ^1_l	θ^2_l	0	0	0
Pure rotation	0	0	0	m_a	θ^1_a	θ^2_a
Translation str. line	m_l	0	0	m_a	θ^1_a	θ^2_a
Rotation par. axes	m_l	θ^1_l	θ^2_l	m_a	0	0
Planar motion	m_l	θ^1_l	0	m_a	0	0

Pure translations - In this case the x axis of the angular frame is always the null vector and the angular frame cannot be defined. Since the angular frame with respect to the world frame in the first time instant is known, the same initial angular frame is used in each time $t = 1,\dots,N$. The resulting invariants for the angular part are $\{m_a\}_t = \{\theta^1_a\}_t = \{\theta^2_a\}_t = \{0\}_t,\ t = 1,\dots,N$. For pure translations between two consecutive time instants t and $t+1$, it is possible to consider $\hat{\boldsymbol{x}}_{a,t} = \hat{\boldsymbol{x}}_{a,t-1}$ and $\hat{\boldsymbol{y}}_{a,t} = \hat{\boldsymbol{y}}_{a,t-1}$, obtaining that $m_{a,t} = 0$.

Pure rotations - In this case the x axis of the linear frame is always the null vector and the linear frame cannot be defined. Since the linear frame wrt the world frame in the first instant is known, the same initial linear frame is used in each time $t = 1,\dots,N$. The resulting invariants for the linear part are $\{m_l\}_t = \{\theta_l^1\}_t = \{\theta_l^2\}_t = \{0\}_t,\ t = 1,\dots,N$. For pure rotations between two consecutive time instants t and $t+1$, it is possible to consider $\hat{\boldsymbol{x}}_{l,t} = \hat{\boldsymbol{x}}_{l,t-1}$ and $\hat{\boldsymbol{y}}_{l,t} = \hat{\boldsymbol{y}}_{l,t-1}$, obtaining that $m_{l,t} = 0$.

Translations along a straight line - In this case the x axes of the linear frame are aligned in all the time instants, and the y axes cannot be defined. Since the linear frame with respect to the world frame in the first instant is known, it is possible to consider $\hat{\boldsymbol{y}}_{l,1} = \hat{\boldsymbol{y}}_{l,2} = \ldots = \hat{\boldsymbol{y}}_{l,N}$. The resulting invariants are $\{\theta_l^1\}_t = \{\theta_l^2\}_t = \{0\}_t,\ t = 1,\dots,N$. For translations along a straight line in two consecutive instants t and $t+1$, it is possible to consider $\hat{\boldsymbol{y}}_{l,t} = \hat{\boldsymbol{y}}_{l,t-1}$, obtaining that $\theta_{l,t}^1 = \theta_{l,t}^2 = 0$.

Rotations about parallel axes - In this case the x axes of the angular frame are parallel in all the time instants, and the y axes cannot be defined. Since the angular frame with respect to the world frame in the first instant is known, it is possible to consider $\hat{\boldsymbol{y}}_{a,1} = \hat{\boldsymbol{y}}_{a,2} = \ldots = \hat{\boldsymbol{y}}_{a,N}$. The resulting invariants are $\{\theta_a^1\}_t = \{\theta_a^2\}_t = \{0\}_t,\ t = 1,\dots,N$. For rotations about parallel axes in two consecutive instants t and $t+1$, it is possible to consider $\hat{\boldsymbol{y}}_{a,t} = \hat{\boldsymbol{y}}_{a,t-1}$, obtaining that $\theta_{a,t}^1 = \theta_{a,t}^2 = 0$.

Planar motions - In this case the rigid body translates on a plane π and rotates about an axis orthogonal to π. By construction, the y axes of the linear frame are orthogonal to π (and parallel to each other) for all $t = 1,\dots,N$. As a consequence, it holds that $\{\theta_l^2\}_t = \{0\}_t,\ t = 1,\dots,N$. The angular motion is a rotation about parallel axes (orthogonal to π), resulting in $\{\theta_a^1\}_t = \{\theta_a^2\}_t = \{0\}_t,\ t = 1,\dots,N$.

3.2.3. Results in motion recognition

The goal of this experiments is to verify the effectiveness of invariant motion descriptors in gesture recognition problems. In order to test the effectiveness of a given feature vector, it is common to use a state-of-the-art classifier and to show an eventual improvement in the classification performance. Hence, in this experiments, the standard k-Nearest Neighbour (k-NN) classifier [16] is exploited. k-NN assigns the query gesture to the class most common among its k nearest neighbors, and it does not require a training step (non-parametric). The number k of nearest neighbors and the distance metrics between features vectors are tunable parameters. This work adopts the Euclidean distance as a metrics and presents results for different values of k.

Three datasets are employed to test the recognition performance of DHB, and to compare the DHB representation with state-of-the-art invariant descriptors. The main characteristics of each dataset are listed in Table 3.2. The English letters dataset includes five gestures performed by the same user. Despite its simplicity, the English letters dataset is useful to understand how to apply invariant representations in gesture recognition problems. The second dataset, namely the pouring drink dataset, consists of five gestures performed by five different users. The presence of multiple users makes the pouring dataset more challenging that the English letters one. The last dataset is the Microsoft research Action 3D dataset [3], a publicly available dataset that consists of twenty actions performed by ten different users. The relatively big number of actions/users makes this dataset a challenging benchmark for gesture recognition algorithms. Results presented in this section are obtained with velocity-based invariants. Indeed, position- and velocity-based invariants are practically the same (see Section 3.2.2) and have the same performance in terms of gesture recognition. The recognition performance of DHB is compared with the performance of DS (De Schutter) [42] and EFS (Extrended Frenet–Serret) [155] representations.

English letters dataset

The goal of this experiment is to compare the recognition performance of invariant and non-invariant motion descriptors. To this end, a simple dataset is considered that includes the five capital letters A, M, N, O, and X from the English alphabet [1]. Each letter is drawn ten times by

Table 3.2.: Datasets characteristics.

Dataset	Actions (#)	Users (#)	Repetitions (#)	Body parts (#)	Sampling rate [Hz]
English letters	5	1	50	1	30
Pouring drink	5	5	10	1	120
MSR Action3D	20	10	3	20	30

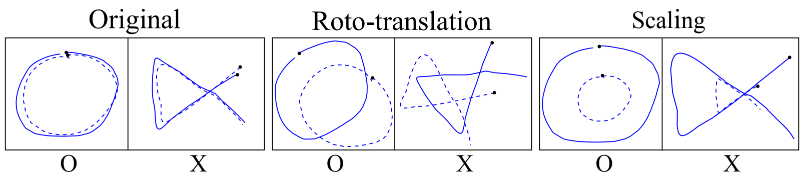


Figure 3.14.: Demonstrations of the letters O and X from the English letters dataset, visualized in the $x-y$ plane (left). Roto–translated (middle) and scaled (right) versions of the original demonstrations. Black points indicate the starting position of each trajectory.

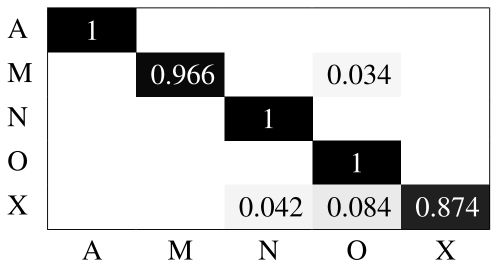
the same user. Motion trajectories are collected by tracking the user's right hand position at 30 Hz with a RGB-D camera. Repetitions of the letters O and X are qualitatively shown in Figure 3.14. In order to show the benefits of the invariance to motion variations, the dataset is augmented with 225 extra demonstrations, generated by applying random affine transformations to the original data, as shown in Figure 3.14. The random transformations produce 40 additional demonstrations per letter. Hence, the final dataset reaches a total of 250 demonstrations (50 demonstrations for each of the 5 letters). This procedure aims at simulating a realistic scenario where the same gesture can be observed or executed from different perspectives. Time derivatives are numerically computed with the sampling time $\Delta t = 1/30$ s, the same frame rate at which data are collected.

A leave-half-out cross validation approach, where a half of the gestures are used for training and the rest for testing, is employed to validate different motion representations. Training and test sets are randomly selected 100 times. Average results, obtained for different values of k (1-NN, 3-NN and 5-NN) with filtered and unfiltered data, are listed in Table 3.3. Confusion matrices for DHB and DS/EFS representations are shown in Figure 3.15. The confusion matrices are obtained with 1-NN and filtered data. Results in Table 3.3 and Figure 3.15 are obtained considering all the invariant values of each representation. Note that the hand orientation is not present in the dataset. Hence, each gesture consists of a pure translation for which EFS and DS become exactly the same representation (see Appendix B.1.2 for further details).

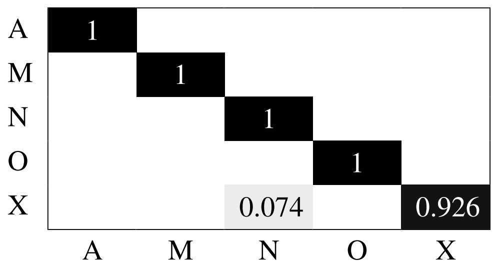
Results in Table 3.3 show that the Cartesian twist has the worst recognition rate. This is because the Cartesian twist is invariant only to translations. The recognition performance of DS/EFS significantly improves when the data are filtered with a moving average filter (window size $w = 5$).

Table 3.3.: Recognition results on the English letters dataset.

	Unfiltered data			Filtered data		
	1-NN	3-NN	5-NN	1-NN	3-NN	5-NN
Twist	78.8%	72.7%	69.6%	79.3%	74.1%	70.4%
DS / EFS	79.56%	79.24%	78.92%	96.96%	96.44%	96.36%
DHB	**94.76%**	**94.36%**	**94.32%**	**98.52%**	**98.32%**	**98.24%**



(a) DS/EFS invariants



(b) DHB invariants

Figure 3.15.: Confusion matrices for the English letters dataset obtained with filtered data and $k = 1$.

The reason is that DS/EFS are ill-conditioned on noisy data, since they require the numerical computation of high-order time derivatives. The performance of DS/EFS descriptors can be further improved by exploiting more sophisticated, off-line filtering techniques like the linear Kalman smoother [127]. However, the usage of off-line filtering techniques limits the applicability of invariant representations in on-line gesture recognition problems. Moreover, off-line filtering algorithms are computationally more expensive than on-line approaches. DHB invariants have the best recognition performance in all considered cases, which implies that DHB invariants are robust to noise in the data.

Pouring a drink dataset

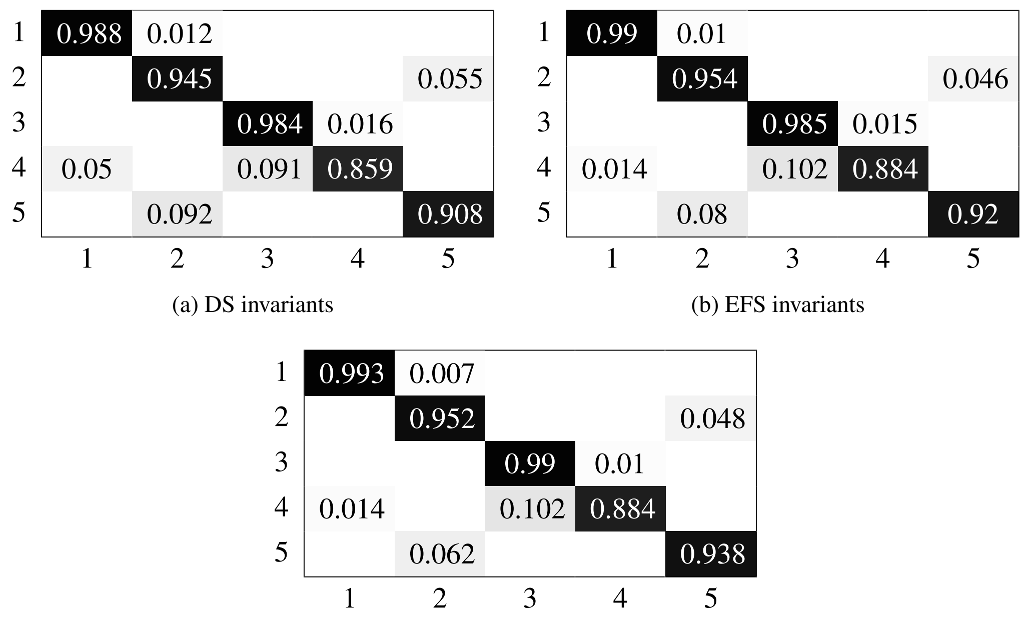
The aim of this experiment is to test the recognition performance of invariant representations among different users. To this end, the adopted dataset consists of five actions—*pour*, *release*, *start*, *stop*, and *take*—performed ten times by five different users (see Table 3.2). Gestures are performed with the right hand. The data collection protocol is as follows. An expert user demonstrates once a gesture, and then asks the subject to repeat the shown action ten times in a row. The subject is also asked to arbitrarily change its pose after each repetition. No particular instruction is given to the user regarding the motion stile or the speed of execution. A Xsens MVN motion capture suit [8] is used to collect Cartesian twists of the user right hand. Data are collected at 120 Hz. The origin of the inertial sensor attached to the right hand is used as reference point. Time derivatives are numerically computed with the sampling time $\Delta t = 1/120\,\mathrm{s}$, the same frame rate at which data are collected. Collected twist trajectories are filtered using a moving average filter ($w = 5$). This is because, as shown in the previous experiment, all representations perform better with filtered data.

A leave-half-out cross validation approach, which uses half of the repetitions of each user as training set and the rest as test set, is exploited to test the performance of DHB, DS, and EFS. Training and test sets are randomly selected 100 times. Average recognition rates for different values of k are reported in Table 3.4. Results in Table 3.4 show that, when all the six invariant values are used, the DS descriptor has poor recognition performance compared to DHB and EFS invariants. As discussed in Appendix B.2, in fact, the DS representation has an increased noise sensitivity due to the projection of linear velocities (and their time derivatives) along the instantaneous screw axis.

Two of the invariants in EFS and DS representations depends on linear and angular jerks (see Appendix B). In particular, the jerk depended invariants for for EFS are e_v^3 and e_ω^3, while for DS are d_v^3 and d_ω^3. As shown in Table 3.4 and Figure 3.16, removing jerk dependent invariants is beneficial

Table 3.4.: Recognition results on the pouring a drink dataset (filtered data).

	All invariants			Without jerk dependent invariants		
	1-NN	3-NN	5-NN	1-NN	3-NN	5-NN
DS	77.45%	80.38%	80.59%	93.66%	92.72%	91.06%
EFS	94.02%	92.86%	91.5%	94.66%	**93.24%**	**91.95%**
DHB	**95.14%**	**93.45%**	**92.08%**	**94.89%**	93.22%	91.9%



(a) DS invariants

	1	2	3	4	5
1	0.988	0.012			
2		0.945			0.055
3			0.984	0.016	
4	0.05		0.091	0.859	
5		0.092			0.908

(b) EFS invariants

	1	2	3	4	5
1	0.99	0.01			
2		0.954			0.046
3			0.985	0.015	
4	0.014		0.102	0.884	
5		0.08			0.92

(c) DHB invariants

	1	2	3	4	5
1	0.993	0.007			
2		0.952			0.048
3			0.99	0.01	
4	0.014		0.102	0.884	
5		0.062			0.938

Figure 3.16.: Confusion matrices for the pouring a drink dataset. Results are obtained with $k = 1$ and without considering jerk dependent invariants in DS and EFS representations.

especially for DS representation. Indeed, the recognition rate of DS without jerk dependent values significantly increases. Results in Table 3.4 allow to conclude that jerk depended values of DS descriptor degrade the performance, and that better recognition performances are achieved by employing a subset of the DS invariant values. Regarding EFS invariants, the performances with and without jerk depended values are comparable. However, the recognition rate slightly increases when a subset of the EFS invariants is considered. On the contrary, removing values from the DHB representation reduces the performance. As shown in Table 3.4, the recognition rate decreases by removing the values θ_v^2 and θ_ω^2 from DHB representation. Note that the values θ_v^2 and θ_ω^2 are removed because they describe similar quantities as e_v^3 and e_ω^3 in EFS. Being the performance of DHB slightly improved when all the invariant values are considered, it is possible to conclude that all the six values in DHB representation contribute to make the motion more distinctive.

MicroSoft Research (MSR) Action3D dataset

This experiment aims at testing the recognition performance of DHB invariants in a more challenging scenario. To this end, the MSR Action3D dataset [3] is used. The MSR Action 3D dataset

Table 3.5.: MSR action Dataset and Recognition Protocol

MSR Action3D dataset: Twenty actions are performed 3 times by 10 users. The 20 actions are highArmWave, horizontalArmWave, hammer, handCatch, forwardPunch, highThrow, drawX, drawTick, drawCircle, handClap, twoHandWave, sideBoxing, bend, forwardKick, sideKick, jogging, tennisSwing, tennisServe, golfSwing, and pickUpThrow.

Action Subsets: The MSR Action3D dataset is split into three action sets (AS1, AS2, and AS3).

AS1	AS2	AS3
horizontalArmWave	highArmWave	highThrow
hammer	handCatch	forwardKick
forwardPunch	drawX	sideKick
highThrow	drawTick	jogging
handClap	drawCircle	tennisSwing
bend	twoHandWave	tennisServe
tennisServe	sideBoxing	golfSwing
pickUpThrow	forwardKick	pickUpThrow

Cross validations: Three tests (T1, T2, and T3) are performed on each action set. In T1, one demonstration for each user, i.e. $1/3$ of the samples, is used for training and the others for testing. In T2, $2/3$ of the samples are used for training and the rest for testing. In T3, demonstrations from half of the users are used for training and the rest for testing. In T1 and T2 data from all the subjects are considered for the training, while T3 is a cross subject test.

contains the repetitions of 20 actions performed 3 times by 10 users. The different actions in the dataset are listed in Table 3.5. A RGB-D camera is used to track the position of 20 parts of the human body at 30 Hz. The experimental protocol described in [165] and summarized in Table 3.5 is used to test the recognition performance of DHB, DS, and EFS. For each gesture in the dataset, the invariant representation of each body part is computed. The origin of the frame attached to each body part is used as reference point. As in the previous experiments, trajectories are filtered with a moving average filter ($w = 5$) before computing the invariants. Time derivatives of the position are computed by numerical differentiation with sampling time $\Delta t = 1/30\,\mathrm{s}$. The results in [135] and [157] show that full-body motions are better recognized when body parts not involved in the movement are discarded. In this experiment, unused body parts are automatically cut-off by applying a thresholding method. A body part P is considered relevant to the current action if P is moved more than $t_r = 0.15\,\mathrm{m}$, where the value of t_r is chosen empirically. The invariants relative to irrelevant body parts are simply zeroed.

Average recognition rates in Table 3.6 are obtained with different motion representations by randomly selecting 100 times the training and test sets. Results for DHB and DS/EFS are ob-

Table 3.6.: Recognition results of DHB, DS/EFS, HOJ3D [165] and Li et al. [94] on the MSR Action3D dataset.

	T1				T2				T3			
	AS1	AS2	AS3	Average	AS1	AS2	AS3	Average	AS1	AS2	AS3	Average
DS/EFS	89.17%	89.17%	98.58%	92.31%	88.51%	88.73%	97.17%	91.47%	87.92%	86.85%	98.22%	91.0%
DHB	97.65%	91.84%	**100%**	**96.49%**	97.96%	93.74%	**100%**	**97.23%**	**96.42%**	**91.3%**	**99.98%**	**95.9%**
HOJ3D	**98.47%**	**96.67%**	93.47%	96.2%	**98.61%**	**97.92%**	94.93%	97.15%	87.98%	85.48%	63.46%	78.97%
Li et al.	89.5%	89.0%	96.3%	91.6%	93.4%	92.9%	96.3%	94.2%	72.9%	71.9%	79.2%	74.7%

tained using a 1-NN classifier and considering all three invariant values for each joint. It is worth considering that only three invariants exists for each joint since the orientation is not stored in the dataset. The unidirectional HOJ3D approach in [165] performs slightly better than DHB invariants in test T1/T2 on AS1/AS2. The reason is that the histograms in HOJ3D contain information about the motion direction, stored into a temporal sequence of joint locations casted into spatial bins. DHB, DS, and EFS invariants, instead, do not contain information about the directionality of the motion, which is useful to discriminate actions in AS1 and AS2. To better understand this point, consider that the actions in AS1 and AS2 are mainly performed with the right arm, and that some actions consist of similar movements executed in different directions. More specifically, experiments show that using DHB invariants the action forwardPunch in AS1 is confused with highThrow, while the actions horizontalArmWave and drawX in AS2 are confused with handCatch and drawCircle respectively. Misclassified actions are performed by moving the right arm in a similar way, but in different Cartesian directions (consider, for instance, forwardPunch and highThrow), and are hard to recognize without considering information about the motion direction. A possible solution to improve the recognition accuracy for similar motions consists in augmenting the descriptor with information about the directionality of the motion, like the normalized velocity in each direction [49]. DHB invariants have, on average, the highest recognition rate in all the considered cases. Moreover, the DHB representation has the highest recognition rate on AS3 and in the cross subject test T3, showing good generalization capabilities among different users.

3.2.4. Results in motion reproduction

The aim of the experiments in this section is to evaluate the generalization capabilities of the reconstruction procedure in Section 3.2.2. To this end, four different experiments are presented. The first experiment shows how different instances of a Cartesian motion are generated from the same DHB representation. The second experiment focuses on transforming a human demonstration into a feasible trajectory for the NAO humanoid robot [4]. In the third experiment a complex task, consisting of three different actions, is effectively executed by a real robot. The last experiment shows how full-body motions are generated starting from the invariant trajectory of each body part.

Multiple trajectories from the same descriptor

Different affine transformed instances of a Cartesian trajectory can be generated by the same DHB descriptor. This result is obtained by reconstructing the Cartesian motion in an arbitrary reference frame and with an arbitrary scaling factor. As a proof of concepts, this experiment employs one demonstration of the letter M in Section 3.2.3. The demonstration is the orange curve in Figure 3.17. Given the Cartesian demonstration, the corresponding DHB descriptor is computed and used to generate affine transformed instances of the original trajectory. In Figure 3.17(a), the motion is reconstructed in a translated ($\boldsymbol{t} = [-0.05, 0.5, 0]^{\mathrm{T}}$ m) initial frame. The same translation $\boldsymbol{t}$ is applied in Figure 3.17(b), together with a scaling factor $s = 0.5$. The initial frame is rotated of -30 deg around the z axis in Figure 3.17(c). The the black dashed lines in Figure 3.17 represent the correspondences (optimal non-linear match) between the original demonstration and the generated trajectories, computed by using the multi dimensional dynamic time warping (MD-DTW) algorithm in [132].

English letters reproduction

This experiment illustrates how to map human hand motion into a robot hand motion by generating scaled motion variants from the same DHB descriptor. The humanoid robot NAO [4] is employed

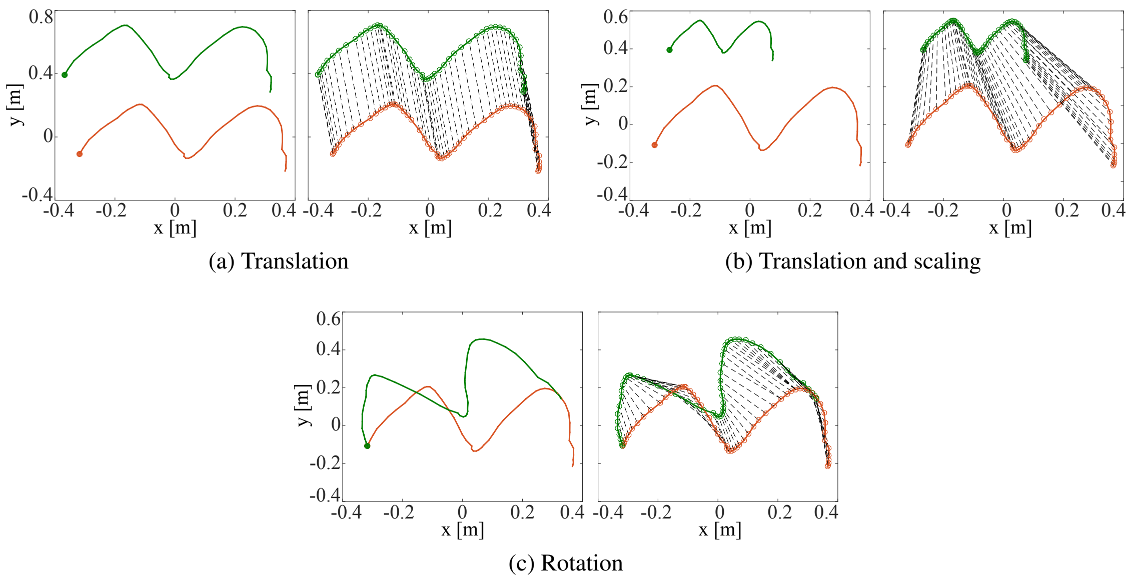


Figure 3.17.: The same DHB descriptor is used to generate affine transformed instances (green solid lines) of the letter M. The orange solid line is the original letter. Black dashed lines are the MD-DTW alignment between original and generated trajectories. For a better visualization, trajectories are plotted in the $x-y$ plane.

to reproduce a demonstration of the letters A and N in Section 3.2.3. The robot performs the motion with the right hand, as shown in Figure 3.18. The human demonstration is expressed in the RGB-D camera frame, which is unknown in this case. However, the DHB representation can directly generate the Cartesian trajectory in the robot reference frame without any processing of the DHB invariants. It is clear, for example considering the arm length, that the NAO robot and the human demonstrator have different kinematics. For this reason, the Cartesian trajectory reconstructed considering the "human" linear scale s_{hum} in (3.61) is unfeasible for the robot. Comparing the maximum velocity of the robot hand, as well as its arm length, with the velocity required to execute the original motion, a suitable trajectory for NAO is generated considering a scaling factor $s_{nao} = (1/12)s_{hum}$. Results of this procedure are shown in Figure 3.18. Reconstructed velocities are transformed into joint angle trajectories with an inverse kinematics algorithms (see Appendix A.2.2) and sent to the robot at 30 Hz.

An automatic procedure can be implemented to find a suitable scaling s_{nao}. If the physical scaling between robot and human is roughly known, it is a good starting point. Otherwise, if the physical scaling is unknown, the procedure is initialized with $s_{nao} = s_{hum}$. The current s_{nao} is used to generate desired velocities and positions. The generated motion is checked to determine if robot constraints are violated. In case of constraint violations, the scaling s_{nao} is reduced (for example halved) and the procedure repeated until no violation occurs. This procedure can be also used to transfer the same motion to different robots.

Pouring a drink

This experiment shows how different DHB descriptors are combined to execute a task consisting of a sequence of elementary actions. To this end, the task of pouring a drink in a cup is considered. The three elementary actions composing the task are *take* the bottle, *pour* the liquid in a cup, and *release* the bottle. The actions were originally demonstrated by a human (see Section 3.2.3) and

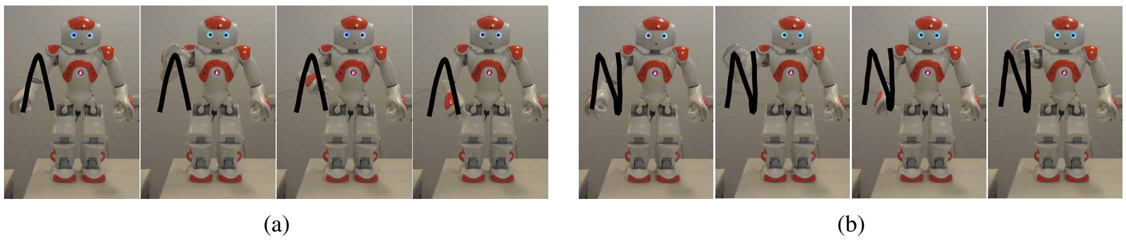


Figure 3.18.: Snapshots of the small humanoids robot NAO that reproduces the letters A and N.

reproduced by a Kuka light-weight (LWR) manipulator [139]. The DHB representation of each action (first demonstration from the first user) is used as motion descriptor. As for the previous experiments, the invariance to roto–translations of DHB representation is exploited to generate the same Cartesian trajectory from different initial configurations. The end-effector pose at the beginning of each action is used to reconstruct on-line the desired velocities, allowing a sequential execution of the three actions without jumps. The results of this procedure are shown in Figure 3.19, where the robot effectively executes the pouring task.

The desired Cartesian pose, computed by numerically integrating the reconstructed velocity, is sent to the KUKA LWR through the fast research interface [139]. The robot controller runs at 1 KHz, while the demonstrations are sampled at 120 Hz, which is the maximum sampling rate of the Xsens MVN motion capture suit. This time difference is compensated by linearly interpolating the desired trajectory. In particular, thirty extra samples are added in each time interval to match the maximum velocity increment that the robot allows. Figure 3.19 shows the case in which the "human" scaling factor $s_{lwr} = s_{hum}$ and the current pose of the end-effector are suitable to achieve the task. It is worth noticing that, being orientation needed to execute the pouring task, the approach in [164] does not apply in this case. Different scaling factors s_{lwr}, as well as the invariance to roto–translations, can be exploited to reach the bottle (cup) placed in different locations. However, this generalization requires a specialized system that computes a suitable affine transformation by recognizing and tracking the bottle (cup) using an external camera sensor. Developing such a system is beyond the scope of this thesis and left as a future work.

An accurate reproduction of the demonstrated motion is of importance in a pouring task. For

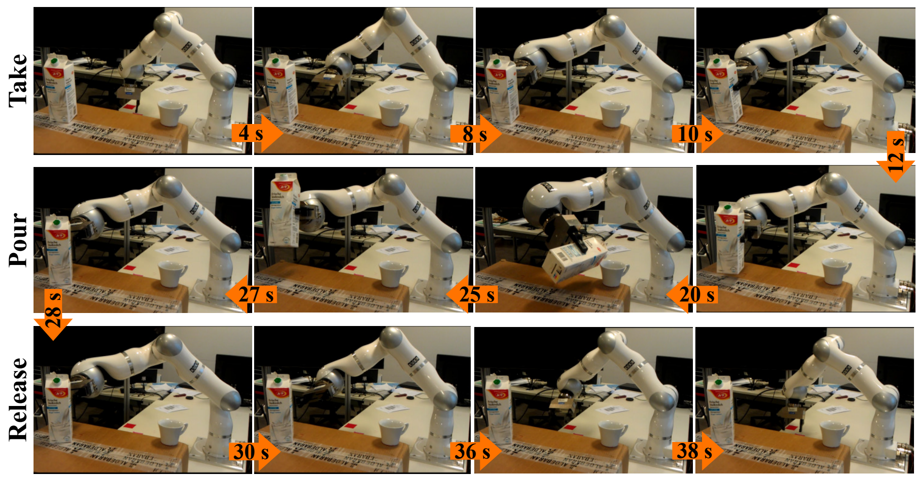


Figure 3.19.: The KUKA LWR executes a pouring task. The task is composed by three consecutive actions: take, pour, and release.

instance, the robot can miss the bottle if the trajectory is not accurately generated. For this reason, it is interesting to compare the reconstruction errors of different invariant representations. The action *take* is considered as an example. For simplicity, the motion is reconstructed in the the same world frame used to collect the data, and the orientation is neglected. The accuracy is measured by calculating the distance between the final reconstructed position and the one in the demonstration. A smaller distance indicates a more accurate motion generation. Three filtering techniques are tested, namely the moving average filter, the anisotropic diffusion [116], and the linear Kalman smoother [127]. The action consists of 295 velocities sampled at 120 Hz.

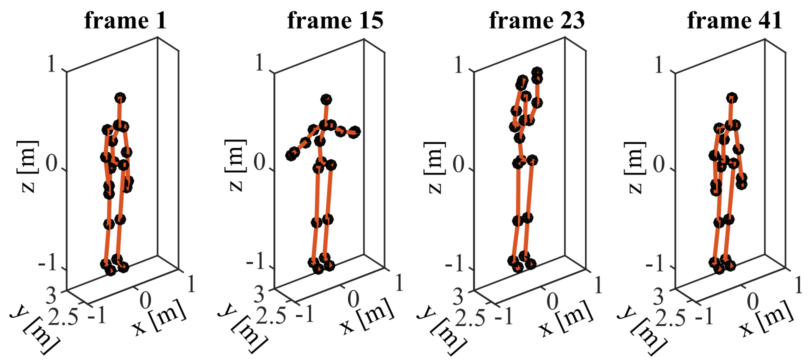
In all the considered cases, DHB achieves the highest accuracy (see the results in Table 3.7). Interestingly, the reconstruction error of DHB is almost independent on the used filtering techniques. This is another empirical proof of the numerical robustness and reduced noise sensitivity of DHB representation (see Appendix B.2 for further details). Regarding DS and EFS invariants, results show that the moving average filter is not suitable for accurate motion reconstruction. Even with a window of 100 samples, the error is still 141 mm. The anisotropic diffusion, which iteratively smooths the trajectory with a Gaussian kernel, is more effective. Iterating 1000 times, the error drops down to 1.15 mm. A Matlab implementation of the anisotropic diffusion takes about 0.4 s to perform 1000 iterations over the 295 samples of the considered action. Hence, the anisotropic diffusion approach cannot be applied on-line. The best result for DS/EFS invariants is obtained with the Kalman smoother, which is an off-line filtering technique.

Table 3.7.: Norm of the reconstruction errors [mm] for the take action obtained with different filtering techniques.

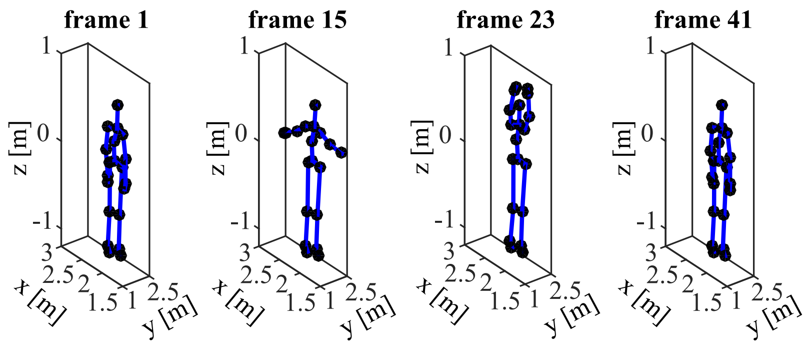
Moving average filter	**DS / EFS**	**DHB**
w=5	464	**2.7e-12**
w=10	191	**3e-12**
w=50	148	**2e-12**
w=100	141	**1.5e-12**
Anisotropic diffusion	**DS / EFS**	**DHB**
i=10	178	**2.7e-12**
i=100	14.5	**3.7e-12**
i=1000	1.15	**2.3e-12**
Linear Kalman smoother	0.56	**2.9e-12**

Full-body motion reproduction

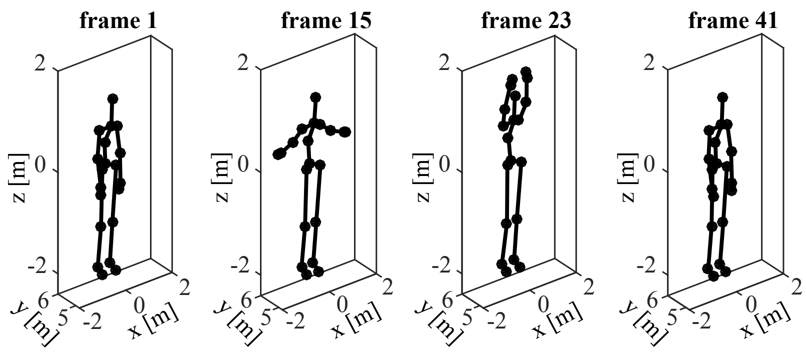
The aim of this experiment is to show that the same DHB representation can be used to reproduce affine transformed instances of a full-body motion. It is know that transfer full-body motions from a human to a humanoid robot is more complicated than reproducing the motion of a single rigid body. Indeed, the kinematic of the human and the robot can vary significantly. Moreover, the mapping has also to consider the dynamics and balancing of the humanoid robot [63, 64]. Being the transfer of full-body motion beyond the scope of this work, a simplified skeleton model is considered in this experiment. In particular, the same skeleton model used in the MSR Action3D dataset (see Section 3.2.3 and Figure 3.20(a)) is exploited to reproduce affine transformed instances of a full-body motion. Given a motion to reproduce, the position of each body part is directly computed from the relative DHB representation. The results of this procedure are shown in Figure 3.20 for the twoHandWave action of the MSR Action3D dataset. Figure 3.20(a) shows the original action performed by the human. The action in Figure 3.20(b) is obtained by applying to the reference frame a rotation of 45 deg along the z axis. This rotation can be interpreted as a rotation of the human around the torso frame. The action in Figure 3.20(c), instead, is reconstructed from DHB invariants with a scaling factor of $s = 2$ (twice the original one) applied to all the body parts. Note that different scaling factors for each body part can be used to reproduce the same motion on two skeleton models with different lengths.



(a) Original action.



(b) Reconstructed action - Rotation (45 deg around z).



(c) Reconstructed action - Scaling ($s = 2$).

Figure 3.20.: Affine transformed instances of a full-body motion (twoHandWave) are generated from the same DHB descriptor. The black bullets indicates the 20 body parts constituting the skeletal model.

3.3. Representation of structured tasks

Previous sections presented effective approaches useful to represent elementary actions or motion primitives. This section, instead, discusses the representation of structured tasks. First, structured tasks are qualitatively defined. The abstract description helps to understand the fundamental aspects of a structured task. The second part of the section presents a formal definition of a structured tasks and discusses the most suited data structure for structured tasks representation.

3.3.1. A qualitative definition of structured tasks

In this work, the term "structured task" is used to indicate a task which involves the execution of multiple elementary actions. The elementary actions include the manipulation of single or multiple objects. Typical examples of structured tasks include preparing a certain recipe or assembly several parts into a final product. A structured task can be hierarchically decomposed into different subtasks. For example, the task of pouring a liquid in a cup can be decomposed into three subtasks, namely take the bottle, pour the liquid in the cup, and leave the bottle on the table. Each subtask can be also decomposed into a series of basic actions. For instance, the subtask take the bottle contains the elementary actions reach the cup and grasp the cup. No further decomposition of the elementary actions is possible. For this reason, elementary actions are often referred as *atomic* actions in this work. A structured task has a variable number of levels in its hierarchy. For example, the aforementioned pouring task has three levels: a *root* level, a subtasks level, and the atomic actions level. However, the pouring task can be part of a bigger task, for example the task of prepare a coffee, which has four levels. The task of prepare a coffee can be part of a bigger task, for example the task of prepare the breakfast, which has five levels, and so on. The number of levels can be theoretically infinite, but three or four level are sufficient to represent typical robotic tasks considered in this work (see Section 5.2). Hence, this work considers the structured tasks as *lower-bounded*, meaning that a structured task cannot be decomposed an infinite number of times. In other words, by iteratively decomposing the structured task into lower levels one will reach the atomic actions level in a finite number of iterations.

Some of the subtasks constituting a structured task have to be executed in a coherent manner, meaning that the subtasks have to be executed on certain objects with a particular order. Considering again the pouring task, the robot has to execute first the take the bottle subtask, then the pour the liquid subtask, and finally the leave the bottle subtask. There are also subtasks that can be executed with an arbitrary order. An example is putting milk and sugar in a cup of coffee, where the order of putting milk and coffee has no influence on the final result. In general, the order of execution plays a role in all the level of the hierarchy. There can be tasks which have to be executed with a fixed or an arbitrary order, as well as atomic actions which have to be executed before (or after) other actions.

3.3.2. A quantitative definition of structured tasks

The qualitative description provided in the previous section is useful to grab general aspects of a structured task. The aim of this section is to convert the qualitative description in the previous section in a formal, quantitative definition of structured tasks. A structured task can be hierarchically decomposed into several levels (see Section 3.3.1), which indicates that a structured task is properly represented by a rooted tree. In general, each node in the tree $\mathcal{B}$ represents a *behaviour*. The root node (abstract behavior) is simply a unique label used to distinguish different structured tasks. Going down the hierarchy the behaviors becomes more specialized until the atomic actions are reached. Atomic actions represent the concrete behaviors that the robot has to execute. Indeed, each action is associated with a motion primitive used to generate motor commands for the robot and execute the task. As discussed in Section 3.1, stable dynamical systems are a valid option to represent point-to-point motion primitives.

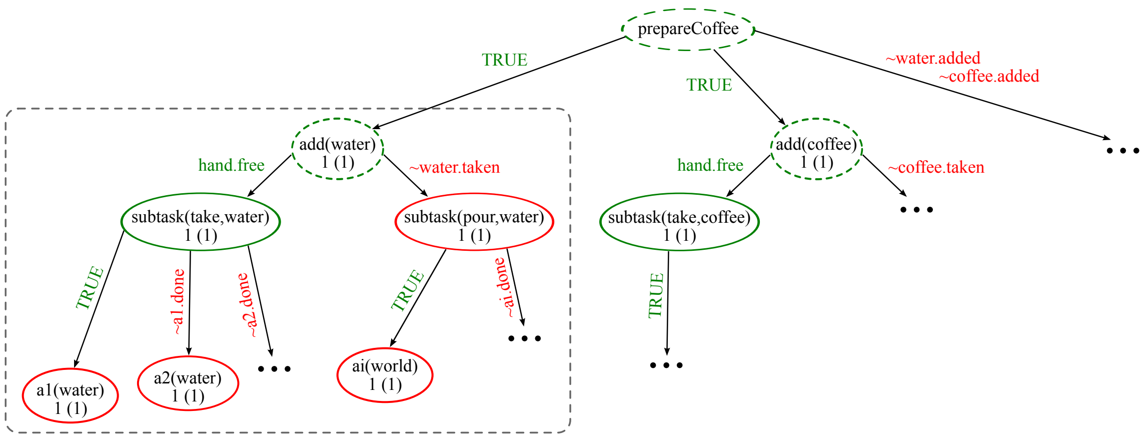


Figure 3.21.: The three used to represent the structured task of prepare a coffee. The three has four levels. *prepareCoffee* is the abstract task, *add(water)* and *add(coffee)* are two concrete tasks, *subtask(take,water)*, *subtask(pour,water)*, and *subtask(take,coffee)* are three subtasks, and *a1(water)*, *a2(water)*, and *ai(world)* are three atomic actions. The gray box highlights the *add(water)* task, which consists of three levels. Green releasers are *true*, red releasers are *false*. For each node, the values outside/inside brackets represent the inverse of the emphasis $1/e_b$ and the magnitude μ_b respectively.

Each node $\mathcal{B}$ is defined by the 6-tuple

$$\mathcal{B} = (m_b, r_b, p_b, x_b, e_b, \mu_b) \tag{3.66}$$

where m_b is a label (name) that identifies the node, r_b is the pre-condition or releaser, p_b represent the post-condition used to check the success of m_b, x_b is the set of sub-behaviors generated by m_b (child nodes), e_b is an emphasis, and μ_b a magnitude value. The emphasis and magnitude values are used to select the next behavior to execute, as detailed in Chapter 4.3.2.

The tree representing the structured task of prepare a coffee is shown in Figure 3.21. The tree has four levels, which is sufficient to represent typical robotic tasks considered in this work (see Section 5.2). The following terminology is used to name the nodes in each levels: abstract task for the root node in the first level, concrete task or task for a node in the second level, subtask for a node in the third level, and atomic action or action for a node in the fourth level. Considering the tree in Figure 3.21, *prepareCoffee* is the abstract task, *add(water)* and *add(coffee)* are (concrete) tasks, *subtask(take,coffee)* is a subtask, and *a2(water)* is an (atomic) action. Note that human friendly labels are used in Figure 3.21 to improve the readability, but the implemented system only requires that labels are uniquely defined. For the root node *prepareCoffee*, the releaser is always *true* and the emphasis is always maximum. The order of execution of *add(water)* and *add(coffee)* is not important in order to prepare a coffee, as underlined by the *true* releaser assigned to both tasks. In order to *add(water)*, the robot first takes the bottle and then pours the water. The robot can grasp the bottle if there are no objects in its hand (releaser *hand.free*), but it cannot pour the water if the bottle is not grasped. This is because the releaser *water.taken* is false. Indeed, *water.taken* and *coffee.taken* are the post-conditions of *subtask(take,water)* and *subtask(take,coffee)* respectively, and they become *true* after the subtask is successfully completed. The atomic actions level is fully learned from human demonstrations by exploiting the approach presented in Section 4.3. As shown in Figure 3.21, the actions belonging to each subtask are executed in the same order they are demonstrated. As a consequence, the first action of each subtask has a *true* releaser.

All other actions in the subtask are constrained to follow the demonstrated order by taking the post-condition of the previous action as pre-condition. For example, the action *a2(water)* has the releaser *a1.done* and it is executed only after *a1(water)* is successfully completed.

3.4. Summary and conclusion

This chapter presented different approaches to represent robotic tasks. The first representation is based on stable dynamical systems (DS). Stable DS are guaranteed to converge towards a given target from any initial state, which makes them particularly suited for representing point-to-point motions. The chapter presented an approach, namely the dynamic movement primitives, useful to represent a single demonstration as a dynamical system. In case multiple task demonstrations are given, a DS can be represented through Gaussian mixture models/regression (GMM/GMR). In general, a GMR based DS is not guaranteed to be globally stable, but it can be stabilized by applying a suitable control input. Notably, this control input is automatically computed in this work given the learned GMM parameters, and it can be smoothly activated/deactivated to balance between the accurate reproduction of the demonstrations and the stability of the DS. This procedure, called contracting Gaussian mixture regression (C-GMR), is theoretically described and empirically compared with other approaches in the literature, showing a good compromise between training time an accuracy. C-GMR accurately reproduces the demonstrations only inside the demonstration area. Outside this area the motion follows an exponentially converging dynamics. This limitation of C-GMR can be potentially overcome by exploiting the full capability of the Contraction theory. In particular, Contraction theory allows to transform the dynamical system by applying a generic coordinate transformation $\boldsymbol{\Theta}(\boldsymbol{x})$. The coordinate transformation $\boldsymbol{\Theta}(\boldsymbol{x})$ can be computed from training data by solving an optimization problem, as proposed in [128]. However, as shown in Section 3.1.4 for the SEDS and SEDSII approaches, the numerical solution of an optimization problem is usually computationally expensive and limits the applicability of the approach in incremental learning scenarios. An interesting future work is to investigate the possibility of finding a suitable coordinate transformation $\boldsymbol{\Theta}(\boldsymbol{x})$ without the need of additional learning steps or optimization.

The second representation maps Cartesian trajectories into a space with known invariance properties, like the invariance to affine transformations. The mapping into an invariant space helps to focus on essential aspects of the motion, making invariant trajectory descriptors effective for motion recognition and reproduction. Two invariant representations are presented, namely a unidirectional representation and a bidirectional one. Since unidirectional representations are not useful for motion reproduction, a particular focus was given to the bidirectional representation, the so-called DHB (Denavit–Hartenberg inspired Bidirectional) representation. Apart from being bidirectional, the DHB representation is a minimal representation (six values) of rigid body motions, it is invariant to affine transformations, time scale, and direction of execution (forward and reverse motions), and it is numerically robust. Those characteristics make DHB an effective motion descriptor, as demonstrated with experiments and comparisons with prominent invariant descriptors. DHB is an effective descriptor for motion generalization. As shown in Section 3.2.4, in fact, affine transformed instances of the same motion are easily generated from the same invariant trajectory. In other words, DHB has high generalization capabilities, being capable of generating the same motion in all the state space of the robot. However, the trajectory generated from an invariant representation is not guaranteed to converge towards a given target. Moreover, DHB is not a compact representation, since an invariant trajectory has only three samples less than a Cartesian one. On the contrary, presented DS based representations are compact and convergent, but they are able to generalize the motion only in a certain area of the state space (around the demonstrations). An interesting research direction consists in combing invariant and dynamical system based representations into a unified approach that combines the benefits of the two original representations.

The last part of the chapter discussed about structured tasks. Structured tasks are firstly qualitatively defined. Intuitively, a structured task is a mission to complete that involves a sequence of actions to perform on certain objects. Actions (objects) are executed (manipulated) arbitrary or by following a predefined order. The qualitative definition in then converted into a formal, quantitative definition. Structured tasks are categorized as a kind of hierarchical structures, specifically a tree, were the root node is the abstract robotic task that is hierarchically decomposed into incrementally more specified behaviors until the last level of the hierarchy is reached. The leaf nodes, in fact, contains the elementary actions which are commanded to the robot in oder to execute the task. Each node in the tree is associated to a set of pre- and post-conditions, as well as to two weighting terms, namely the emphasis and the magnitude. As discussed in Chapter 5, conditions and weights in each node are exploited by an attentional executive system to regulate the task execution. A known problem of tree structures is the scalability. When more and more nodes are added, in fact, the tree search becomes more and more computationally expensive. The problem is alleviated in this work by equipping the attentional system with long term and working (short term) memories. Roughly speaking, the full task structure is stored in the long term memory, while the working memory only expands the nodes required to reach the next action (leaf node) to execute. Formal details of this procedure are given in Chapters 4 and 5. In order to execute the task on real robots, commanded actions are associated with a database of motion primitives, used to generate the robot's trajectory at run time. The problem of learning such a dataset from human demonstrations and associating each motion primitive to the corresponding action in the tree is considered in Chapter 4.

CHAPTER 4

Learning robotic tasks form human demonstrations

Robotic skills acquisition, as well as the skill adaptation to novel, possibly dynamic environments, are of importance for the integration of robotic devices in both industrial and service scenarios. This chapter discusses the problem of transferring skills from humans to robots in a *simple*, *natural*, and *intuitive* manner, allowing people without a robotic or programming background to transfer their skills to the robotic device. Following the programming by demonstration paradigm, the chapter starts investigating the problem of demonstrating novel tasks to a single robotic manipulator (uni-manual task). Then, the problem is considered of transferring a skill to human-like robots with two arms (dual-arm task). Finally, intuitive teaching is exploited to let the robot learning a structured task from human demonstrations.

4.1. Learning uni-manual skills

Kinesthetic teaching is a simple, natural, and intuitive way that humans use to teach new skills. In kinesthetic teaching, a teacher manually guides the partner (robot) during the task execution. During the physical guidance, robot's sensory data, like the position of the end-effector or the joint angles, are stored to create a task demonstration. As discussed in Chapter 3, collected demonstrations are usually represented in a compact form that allows task generalization and reduces memory requirements. Kinesthetic teaching is a well established methodology in the field of programming by demonstration [14], and its benefits and advantages in skill transfer are well recognized [162]. Compared to imitating the human movements (imitation learning), there are three main advantages in physically guiding the robot:

- Collected data are directly used to generate the robot's trajectory. In contrast, motion retargetting is required to imitate human movements, due to the different kinematics between human and robot.
- External motion tracking systems are not required in kinesthetic teaching, since only robot sensors are used to collect data.
- The teacher is sure that the robot is able to reproduce the demonstrated task without violating joint limits.

However, kinesthetic teaching suffers from the following drawbacks:

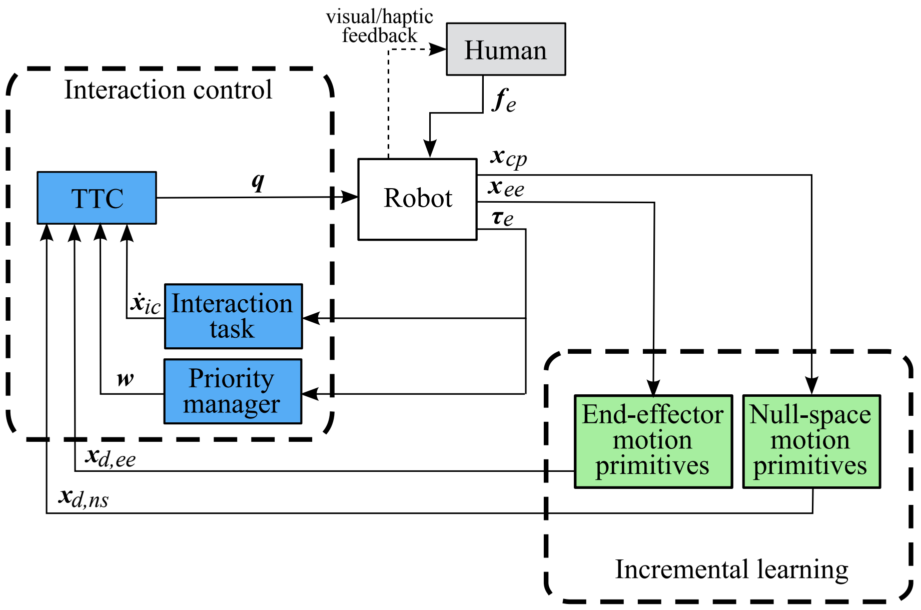


Figure 4.1.: Overview of the framework for incremental kinesthetic teaching of end-effector and null-space motions.

- A specialized controller is required to safely interact with the robot. The simplest solution is to apply the gravity compensation control described in Appendix A.2.3, which works only for robots equipped with force/torque sensors.
- It is hard for the user to handle many, possibly redundant, degrees-of-freedom (DoF), as underlined by the user study in [162].

Considering the presented advantages and disadvantages, it is clear that is hard to kinesthetically teach robots with many DoF like humanoid robots. In this case, imitation learning offers a valid alternative (see Section 4.2). For a single robotic manipulator, instead, kinesthetic teaching often represents the best option.

4.1.1. A framework for incremental kinesthetic teaching

As discussed in [162], users encounter difficulties to handle redundant DoF. However, redundant degrees-of-freedom can be exploited to execute multiple tasks at the same time. Figure 4.1 shows a framework that allows incremental kinesthetic teaching of end-effector and null-space motion primitives. The *incremental learning* block in Figure 4.1 contains end-effector and null-space motion primitives. These motion primitives are learned in batch mode from kinesthetic demonstrations and used reproduce the task. This work exploits hidden Markov models to represents the motion, since they allow incremental learning, but other choices are possible. The approach for motion leaning and generation using HMM is described in Appendix A.3.3 and summarized in Algorithm 3. The motion primitives learned in batch mode may not be appropriate to execute the task in a different scenario. In this case, it is useful to give to the user the possibility of incrementally refining the learned task during the task execution. By kinesthetically demonstrating the task during the execution, in fact, the robot learns naturally coordinated movements of its links [91]. Moreover, the demonstrator can modify only part of the task, leaving the rest unchanged. The incremental refinement requires two ingredients. First, the motion primitive representation has to permit incremental refinement. As detailed in [91] and in Appendix A.3.3, the parameters of a

Algorithm 3 Motion Learning and Generation using HMM

```
Motion primitive learning:
input a motion sequence ${}^{s}\boldsymbol{O}$
 1. Introduce a normalized time variable for each state
 2. Learn HMM parameters $\boldsymbol{\lambda}$ from ${}^{s}\boldsymbol{O}$ using the Baum-Welch algorithm [123]
 3. Compute optimal state sequence $\boldsymbol{Q}^{\star} = \{q(t)\}$ for ${}^{s}\boldsymbol{O}$ using the Viterbi algorithm [123]
 4. Compute the relative temporal sequence ${}^{t}\boldsymbol{O}$ from $\boldsymbol{Q}^{\star}$
 5. Compute ${}^{ts}\boldsymbol{\Sigma}$ for each state from ${}^{s}\boldsymbol{O}$ and ${}^{t}\boldsymbol{O}$
return $\boldsymbol{\lambda} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{c}, {}^{t}\boldsymbol{\mu}, {}^{s}\boldsymbol{\mu}, {}^{tt}\boldsymbol{\Sigma}, {}^{ts}\boldsymbol{\Sigma}, {}^{ss}\boldsymbol{\Sigma}\}$

Motion generation:
input a HMM model $\boldsymbol{\lambda}$
 6. Generate an optimal state sequence $\boldsymbol{Q}$ using (A.26), (A.27)
 7. From $\boldsymbol{Q}$ calculate the relative temporal sequence ${}^{t}\boldsymbol{O}$
 8. From ${}^{t}\boldsymbol{O}$ calculate the responsibilities $\gamma_i$ using (A.28)
 9. Calculate the conditional spatial data ${}^{s}\boldsymbol{o}_i(t)$ using (A.29)
10. Generate the continuous trajectory ${}^{s}\boldsymbol{o}(t)$ using (A.30)
return ${}^{s}\boldsymbol{O}(t) = \{{}^{s}\boldsymbol{o}(t)\}$
```

hidden Markov model can be incrementally refinement as novel demonstrations are provided. Second, the user has to safely interact with the robot to provide additional task demonstrations. Task execution and safe human–robot interaction are considered as two prioritized tasks by the *interaction control* block in Figure 4.1, where the *Task Transition Controller* (TTC) is responsible for the insertion, removal, and transition of these tasks. The motion refinement proceeds in this way. The TTC executes a learned motion primitive. During the execution, the user can modify the learned skill by physically guiding the robot. In case of physical interaction, a new task is generated in the form of a desired velocity $\dot{\boldsymbol{x}}_{ic}$ at the contact point. This separation of the motion primitive and the physical guidance in tasks with different priorities makes possible to incrementally refine both end-effector and null-space motions. The interaction task is initially inserted with the lowest priority. Hence, the robot tries to execute this task using redundant DoF, without affecting the end-effector task execution. In this case, the learned behavior can be considered as a null-space motion primitive. The priority of the interaction task can be increased when it is not correctly executed. Tasks priorities are changed by considering the force that the user is applying on the robot. In particular, if the user perceives that the robot is not accomplishing its guidance, then he increases the applied force. Hence, a threshold on the external force is used to decide the tasks priority. When the user guides the robot touching the end-effector, one between the interaction task and the end-effector motion primitive cannot be executed. At the beginning, the robot keeps following its original motion. Then, the user increase the contact force, the tasks are smoothly switched and the robot can accomplish the physical guidance and refine its behavior. The interaction control block is detailed in the following section.

4.1.2. A customized controller for physical robot guidance

A controller for incremental kinesthetic teaching has to guarantee an accurate motion primitive execution in free motion and the user safety during the physical interaction. In order to realize the incremental motion refinement by physical contact, this work considers task execution and safe human–robot interaction as two prioritized tasks. The priorities of these tasks are not fixed, but they are smoothly and continuously changed during the execution by the task transition control (TTC) [10]. The TTC is a general control framework that allows arbitrary and smooth task transitions. Its main features are described as follows.

Task transition control

A task T of a robot can be defined by a tuple $(\dot{\boldsymbol{x}}, \dot{\boldsymbol{x}}_d)$ in the velocity level where $\dot{\boldsymbol{x}} \triangleq \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$ is the task variable, $\dot{\boldsymbol{x}}_d(\boldsymbol{q},t)$ is the desired trajectory of $\dot{\boldsymbol{x}}$, $\boldsymbol{q}$ is the generalized coordinate of the robot, and $\boldsymbol{J}(\boldsymbol{q})$ is the Jacobian of the manipulator. As discussed in Appendix A.2.2, the inverse kinematics problem consists in finding $\dot{\boldsymbol{q}}$ that minimizes the task error $\|\dot{\mathbf{e}}\| \triangleq \|\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}\|$. In general, the definition of the task is not fixed for whole operation time and multiple tasks $T_1, \cdots, T_k$ can exist on the same time. For example, in this work, the end-effector needs to follow a given motion primitive and, if there is human intervention, the robot has to accomplish the physical guidance. Hence, two different tasks are defined: *i)* the end-effector task T_{ee} and *ii)* the interaction control task T_{ic}. In this work, T_{ic} is an admittance control that transforms external forces into the desired velocity $\dot{\boldsymbol{x}}_{ic}$, as discussed later in this section.

The relation between the two tasks, or their priorities, can be smoothly and continuously changed during the motion execution. For the clearness of the discussion, this work follows the same mathematical convention of a set of tasks proposed in [10]. More specifically, (T_{ee}, T_{ic}) indicates an unprioritized accumulation of two tasks, i.e. T_{ee} and T_{ic} have the same priority. $[T_{ee}, T_{ic}]$ indicates a prioritized accumulation in which T_{ee} has priority. As already mentioned, the priorities of T_{ee} and T_{ic} are changed in each time by the strength of the human intervention. The following rules are applied:

- $T^1 = T_{ee}$. If there is no human intervention, only T_{ee} exists.
- $T^2 = [T_{ee}, T_{ic}]$. If there is a weak human intervention, both tasks exist and T_{ee} has priority.
- $T^3 = [T_{ic}, T_{ee}]$. If there is a strong human intervention, both tasks exist and T_{ic} has priority.

When the task priorities need to be changed during the execution, smooth transitions between tasks T^1, T^2, and T^3 are necessary to prevent discontinuities (jumps) in the joint velocities. The idea of the TTC is to interpolate the joint trajectories directly by using the barycentric coordinates (see Definition 1) and to design smooth transitions of the barycentric coordinates using a linear dynamical system (see Theorem 3). More in details, the inverse kinematic solution $\dot{\boldsymbol{q}}^i$ for each task T^i is calculated separately using the inverse kinematic approach proposed in [9]. The robot's joint velocity is then computed as $\dot{\boldsymbol{q}} = \sum_{i=1}^{l} w^i \dot{\boldsymbol{q}}^i$, where w^i are the barycentric coordinates.

Definition 1 (Barycentric Coordinates)**.** The barycentric coordinates of $\dot{\boldsymbol{q}}$ with respect to $\mathcal{Q} \triangleq \{\dot{\boldsymbol{q}}^1, \cdots, \dot{\boldsymbol{q}}^l\}$ is defined as any set of real coefficients $w^1, \cdots, w^l$ depending on $(\boldsymbol{q},t)$, such that all the following properties hold:

- Nonnegativity: $w^i \geq 0$.
- Linearity: $\dot{\boldsymbol{q}} = \sum_{i=1}^{l} w^i \dot{\boldsymbol{q}}^i$ with $\sum_{i=1}^{l} w^i = 1$.
- Smoothness: $w^i \in C^s(\boldsymbol{q},t)$.

The scalar $s \in \mathbb{N}^{\geq 0}$ depends on the degree of smoothness needed.

Theorem 3 (Task Transition Control)**.** *The TTC given by* (4.1) $\sim$ (4.5) *provides smooth and arbitrary task transitions within* $\mathcal{T} = \{T^1, \cdots, T^l\}$*, as well as bounds the inverse solutions such that*

$\|\dot{\boldsymbol{q}}\| \leq \max\{\|\dot{\boldsymbol{q}}^1\|, \cdots, \|\dot{\boldsymbol{q}}^l\|\}$

$$\dot{\boldsymbol{q}} = \boldsymbol{q}\boldsymbol{w}, \tag{4.1}$$

$$\boldsymbol{w}^{(s+1)} = -\sum_{j=1}^{s} k_j \boldsymbol{w}^{(j)} + k_0(\boldsymbol{w}_d - \boldsymbol{w}), \tag{4.2}$$

$$\boldsymbol{w}_d(\boldsymbol{q}, t) \in \{\hat{\mathbf{e}}^1, \cdots, \hat{\mathbf{e}}^l\} \subset \mathbb{R}^l, \tag{4.3}$$

$$\boldsymbol{w}(t_i) \in \{\boldsymbol{a} \in \mathbb{R}^n : \mathbf{1}^{\mathbf{T}}\mathbf{a} = \mathbf{1},\ \mathbf{a} \geq \mathbf{0}\}, \tag{4.4}$$

$$\boldsymbol{w}^{(j)}(t_i) = \mathbf{0},\ \forall j \in \mathbb{N}_{\leq k}, \tag{4.5}$$

where $\boldsymbol{q} \triangleq [\dot{\boldsymbol{q}}^1 \ \cdots \ \dot{\boldsymbol{q}}^l] \in \mathbb{R}^{n\times l}$, $\boldsymbol{w} \triangleq [w^1 \ \cdots \ w^l]^T \in \mathbb{R}^l$, $\boldsymbol{w}^{(j)} \triangleq d^j\boldsymbol{w}/dt^j$, $s \in \mathbb{N}^{\geq 0}$, $\mathbf{1} \triangleq [1 \ \cdots \ 1]^T \in \mathbb{R}^l$, $\{\hat{\mathbf{e}}^1, \cdots, \hat{\mathbf{e}}^l\}$ *is a set of the standard basis in* $\mathbb{R}^l$, *and* $\{k_0, \cdots, k_k\} \subset \mathbb{R}$ *are stabilizing control gains that don't generate overshoot of the* $(s+1)$*-th order linear system.*

The proof of Theorem 3 can be found in [10]. For the purpose of this work, it is important to consider that arbitrary task transitions and the boundedness of the inverse solutions are guaranteed by the non-negativity and the linearity properties of the barycentric coordinates. Moreover, the smooth task transition is guaranteed by the linear dynamical system that generates the barycentric coordinates. Task transitions are triggered by the discrete input values that correspond to each task T^i. In this work, a threshold f_i is used to insert the interaction control task as the lower priority task, generating extra null-space motions. Another threshold f_s is used to switch the priority between the two tasks, as shown in Table 4.1. The f in Table 4.1 is the norm of the applied external force. The rules in Table 4.1 can be interpreted as follows. The robot firstly tries to project T_{ic} in its null-space. If the human perceives that the task is not correctly executed, he simply applies a bigger force until T_{ic} becomes the first priority task and the robot accomplishes the guidance.

Table 4.1.: Task Transition Rule

$f < f_i$	$f_i \leq f < f_s$	$f_s \leq f$
T_{ee}	$[T_{ee}, T_{ic}]$	$[T_{ic}, T_{ee}]$

Interaction task definition

The task transition control transforms external Cartesian forces in a desired velocity using the relationship

$$\dot{\boldsymbol{x}}_{ic} = c\boldsymbol{f}_e, \tag{4.6}$$

where $\dot{\boldsymbol{x}}_{ic}$ is the velocity of the interaction task, $\boldsymbol{f}_e$ is the applied external force, and c is a tunable gain. It is clear that Equation (4.6) requires an estimation of the external Cartesian force applied to the robot. The estimation of the external force requires two steps. First, the external torque $\boldsymbol{\tau}_e$ applied to each joint have to be estimated. Second, given the contact point C, the external force $\boldsymbol{f}_e$ is computed by inverting the well-known equation $\boldsymbol{\tau}_e = \boldsymbol{J}_C^T\boldsymbol{f}_e$, where $\boldsymbol{J}_C$ is Jacobian of the contact point.

Several approaches have been proposed to estimate the external torque and the link where the contact occurs (contact link). For instance, the work in [58] exploits a momentum based disturbance observer for collision detection and reaction. The work in [130] proposes an approach to estimate the external torque using only encoders. This approach is well-suited for robots that do not have torque sensors in each joint. For the KUKA LWR manipulator, an estimation of the

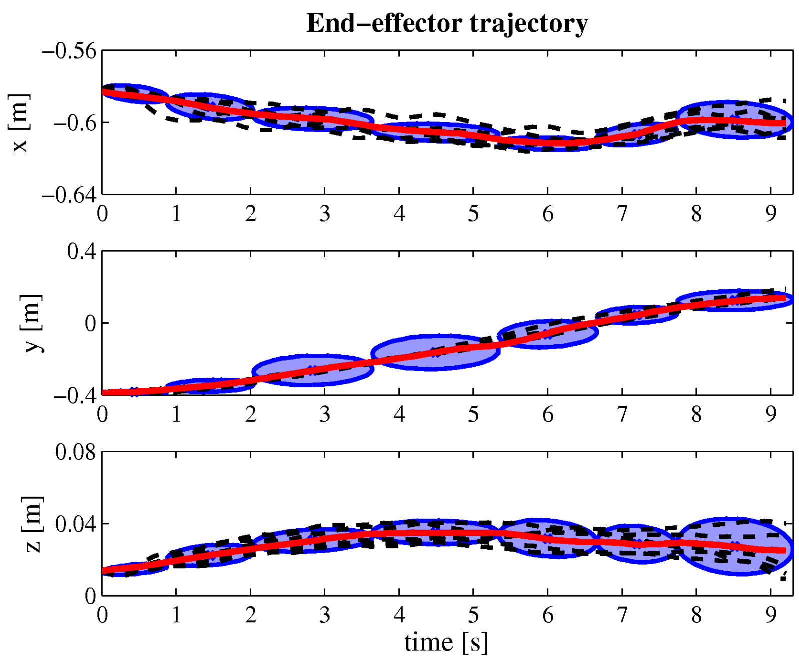


Figure 4.2.: The pick-and-place motion is represented using a left-to-right HMM with seven states and one Gaussian for each state. Five demonstrations (black dashed lines) are used. The blue ellipses represent the learned Gaussian in each state, while red solid lines are smooth trajectories generated by applying Algorithm 3.

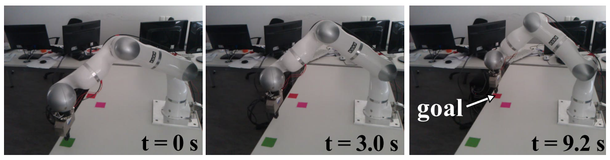
external joint torque is provided through the *Fast Research Interface* [139]. Being the provided torque ideally zero for all the joints located after the contact point, the touched link can be easily determined using a thresholding method. Precise contact point estimation becomes a challenging problem without the usage of an exteroceptive sensor (e.g., cameras) [98]. Since the estimation of the contact point is behind the scopes of this work, it is simply assumed that the contact always occurs at the end of the contact link.

4.1.3. Pick-and-place learning and refinement

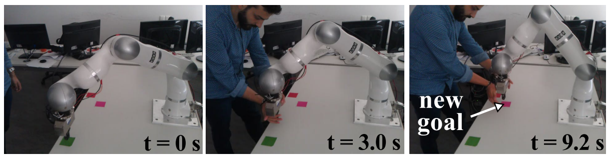
The aim of these experiments is to test the effectiveness of the presented framework for incremental kinesthetic teaching. The end-effector task is the pick-and-place (point-to-point) motion shown in Figure 4.2 and Figure 4.3(a). The pick-and-place motion is learned in batch mode (see Appendix A.3.3) from 5 kinesthetic demonstrations. In order to facilitate the off-line teaching, a gravity compensation controller is used (see Appendix A.2.3). The end-effector orientation is kept constant during the execution of the task. The number of hidden states in the HMM, as well as the number of Gaussians in each state, are empirically chosen to 7 and 1 respectively. These experiments employ the *left-to-rigth* HHM model [123], being this model the best-suited for representing point-to-point motions. The gain $c = 0.005\,\mathrm{m/Ns}$ is used in Equation (4.6) to transform the external forces $\boldsymbol{f}_e$ into velocities.

End-effector motion refinement

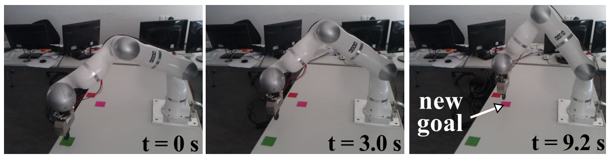
In this experiment, the initial goal position, located at $\boldsymbol{g} = [-0.6, 0.14, 0.027]^{\mathrm{T}}\,\mathrm{m}$, is incrementally refined to reach the new target $\boldsymbol{g}' = [-0.5, 0.006, 0.027]^{\mathrm{T}}\,\mathrm{m}$. During the execution, the user



(a) Original trajectory



(b) Kinesthetic teaching



(c) Refined trajectory

Figure 4.3.: Snapshots of the end-effector motion refinement procedure. (a) The end-effector executes the learned trajectory. (b) The teacher guides the robot toward the new target position. (c) After three iterations, the robot is able to execute the new task.

corrects the trajectory by physically guiding the robot towards the new goal. Snapshots of this procedure are shown in Figure 4.3.

At the beginning of the execution, the end-effector motion is the unique task in the stack. When the user starts to interact with the robot ($\|\boldsymbol{f}_e\| > 5\,\mathrm{N}$), the TTC generates the new task $[T_{ee}, T_{ic}]$. Being the tasks $[T_{ee}]$ and $[T_{ee}, T_{ic}]$ in conflict, the robot cannot accomplish the physical guidance. Hence, the user applies a bigger force and the task priorities are smoothly switched ($\|\boldsymbol{f}_e\| > 15\,\mathrm{N}$). After three repetitions, the robot is able to reach the new target position $\boldsymbol{g}'$, as shown in Figure 4.3(c) and Figure 4.4. The interaction starts at about 3 seconds after starting the execution. Hence, the first 3 seconds of the motion primitive (before the interaction) are left unchanged.

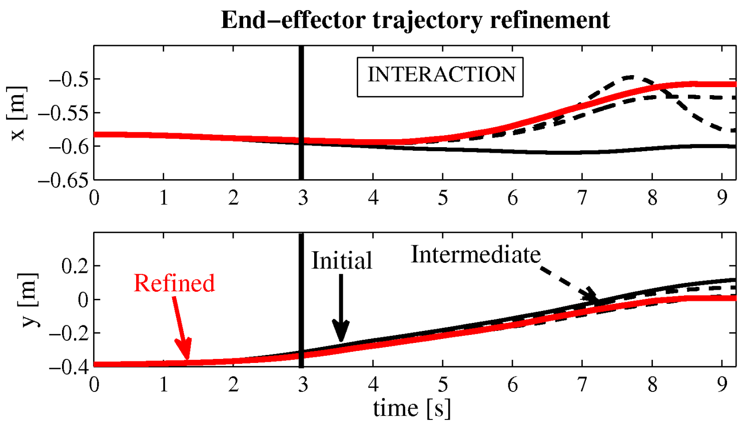
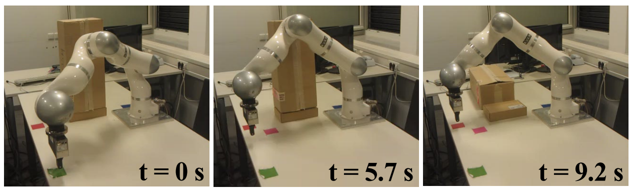
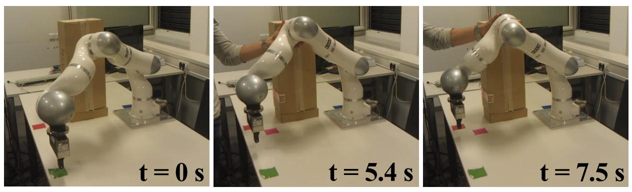


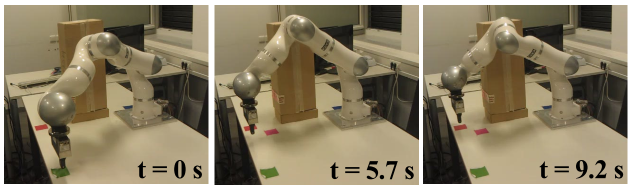
Figure 4.4.: Results of the incremental motion refinement procedure. The robot reaches the new goal position $\boldsymbol{g}' = [-0.5, 0.006, 0.027]^{\mathrm{T}}$ m after three demonstrations. The motion in the z direction is not showed because it is not updated during the kinestethic teaching.



(a) Collision



(b) Kinesthetic teaching



(c) Autonomous execution

Figure 4.5.: The elbow motion refinement procedure. (a) An unforeseen obstacle is placed on the elbow trajectory. (b) The teacher demonstrates a collision-free elbow trajectory. (c) After three iterations, the robot executes both end-effector and null-space tasks.

Elbow motion refinement

As shown in Figure 4.5(a), an unforeseen obstacle is putted along the trajectory of the robot's elbow. The end-effector motion is the pick-and-place task in Figure 4.2. To avoid the collision, the user physically teaches a collision-free path for the robot's elbow (see Figure 4.5(b)). After three iterations, the robot executes the end-effector task while avoiding the collision with the obstacle (see Figure 4.5(c)). Results of the incremental learning procedure are illustrated in Figure 4.6. The interaction begins at about 5.4 s after starting the execution. Hence, the first 5.4 s of the motion primitive (before the interaction) are left unchanged. The thresholds used to insert T_{ic} and to switch the priority are chosen as $f_i = 5\,\mathrm{N}$ and $f_s = 15\,\mathrm{N}$ respectively.

To better understand the role of the used thresholds, three experiments are performed by varying the value of f_s. Recall that the threshold f_s is used to switch the priority between the end-effector task T_{ee} and the interaction task T_{ic}. The threshold used to insert T_{ic} is kept constant at $f_i = 5\,\mathrm{N}$.

$f_i = f_s = 5\,\mathrm{N}$ - With this choice the interaction task T_{ic} is directly inserted as the first priority task, as shown in Figure 4.7(a). In this case the physical guidance is easier because the robot can use more degrees-of-freedom (DoF) to accomplish the guidance. However, if the tasks are in conflict, errors in the motion primitive execution are accumulated. In this specific case, T_{ee} and T_{ic} are not in conflict, since the end-effector position tracking error in Figure 4.7(a) is relatively small (less than 3 mm).

$f_i = 5\,\mathrm{N}$, $f_s = 30\,\mathrm{N}$ - With this choice the T_{ic} is always executed in the null-space of T_{ee}, as shown in Figure 4.7(b). Hence, the robot uses only the redundant DoF to execute T_{ic}. If the robot can accomplish the physical guidance, then the task are not in conflict and the new motion primitive can be effectively executed in the null-space of T_{ee}.

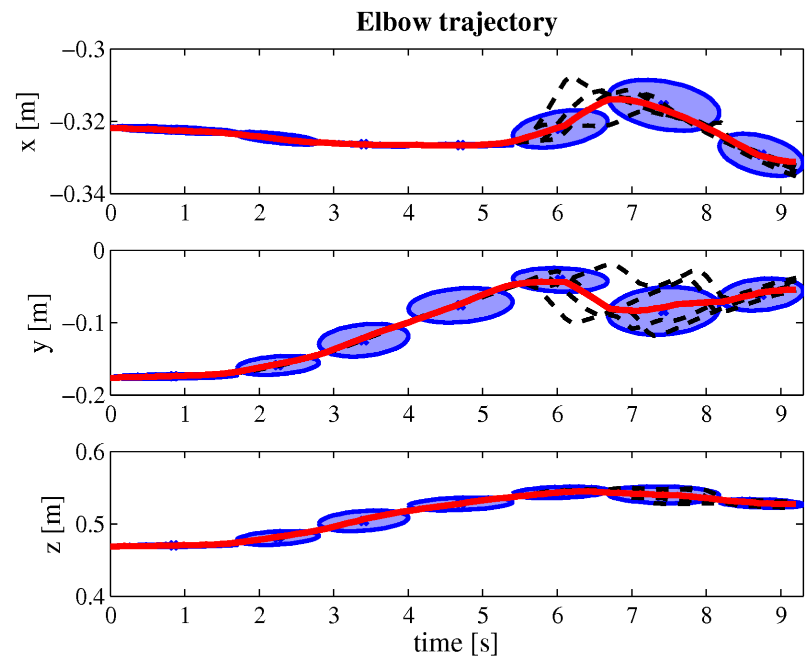


Figure 4.6.: Elbow motion refinement using a HMM with seven states and one Gaussian for each state (blue ellipses). Three demonstrations (black dashed lines) are incrementally given. Red lines are trajectories generated by Algorithm 3.

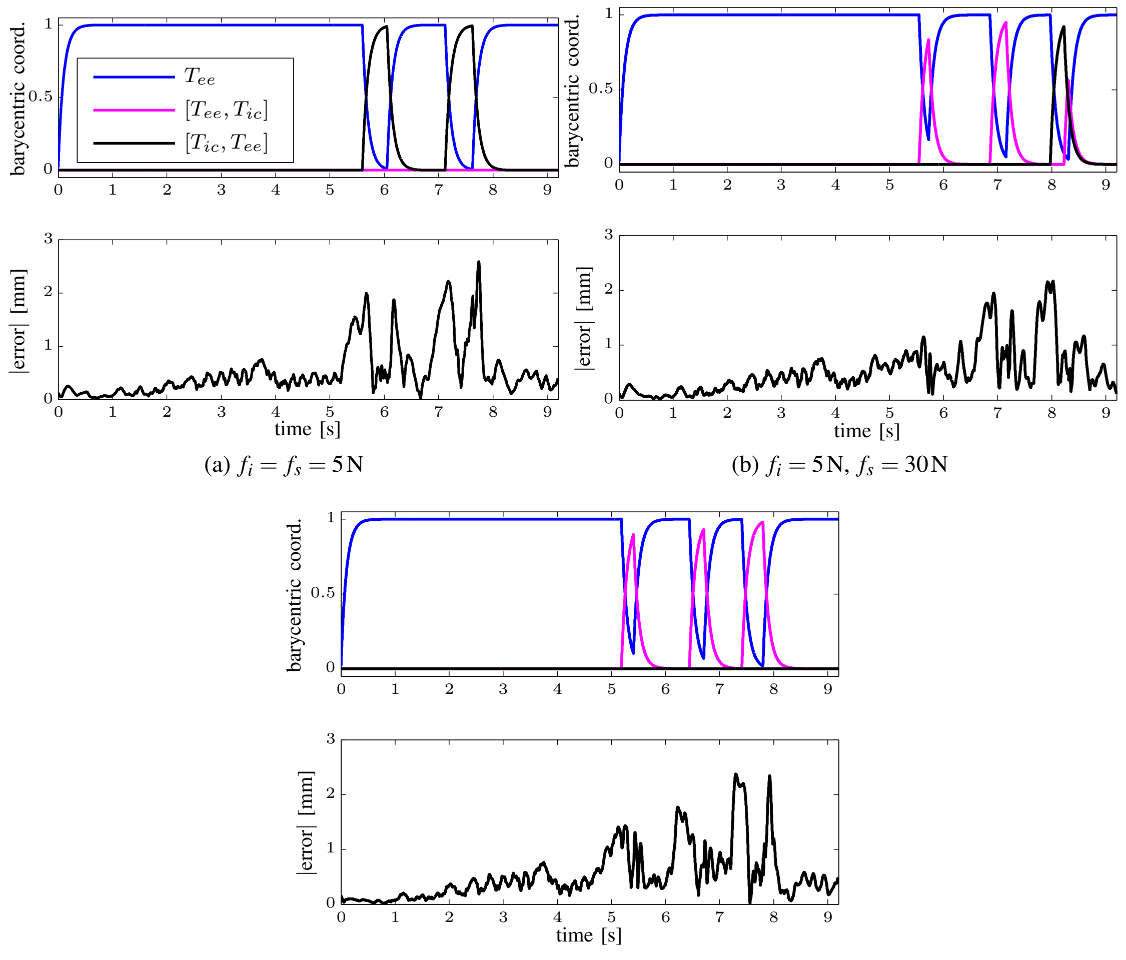
$f_i = 5\,\text{N}$, $f_s = 15\,\text{N}$ - With this choice transitions occur between T_{ic} and T_{ee}, as shown in Figure 4.7(c). These transitions are needed to refine the end-effector motion primitive (T_{ee} and T_{ic} are in conflict). As a general rule, an intermediate value for f_s is always suggested, since it is not know a priori if the physical guidance is in conflict or not with the end-effector motion. During the kinesthetic teaching, the user can figure out if the end-effector deviates significantly from the desired path and, in case, the initial order of priorities has to be reconsidered.

The barycentric coordinates in Figure 4.7 show several transitions between $[T_{ee}]$ and other tasks. This is mainly due to the way the user is teaching the robot. The user, in fact, moves the elbow far away from the obstacle a first time. While the execution proceeds other corrections may be needed, and so on until T_{ee} is completed. The interaction controller (see Section 4.1.2) combines the tracking performances of position controllers with an increased flexibility. Figure 4.8 illustrates the end-effector position tracking errors for position and impedance controllers. The position controller accurately tracks the motion also during the interaction, but it prevents the kinesthetic teaching. On the contrary, to allow physical guidance with impedance control, one has to set low impedance gains[1] penalizing the end-effector task execution also without contacts.

4.2. Learning dual-arm skills

Kinesthetic teaching is effectively exploited in the previous section to transfer skills from a human demonstrator to a robotic manipulator. However, kinesthetic teaching does not represent the optimal approach when the number of degrees-of-freedom is high. This is because it is hard for a single demonstrator to handle many degrees-of-freedom. Considering advantages and disad-

[1]Results in Figure 4.8 are obtained with a stiffness matrix defined as $\boldsymbol{S} = 200\boldsymbol{I}$ N/m and a damping matrix defined as $\boldsymbol{D} = 2d\sqrt{200}\boldsymbol{I}$ sN/m, where $d = 0.7$ to avoid overshoot.



(a) $f_i = f_s = 5\,\text{N}$ (b) $f_i = 5\,\text{N}$, $f_s = 30\,\text{N}$ (c) $f_i = 5\,\text{N}$, $f_s = 15\,\text{N}$

Figure 4.7.: Barycentric coordinates and end-effector position tracking error for different values of f_i and f_s. The interaction controller permits null-space kinesthetic teaching and end-effector task execution.

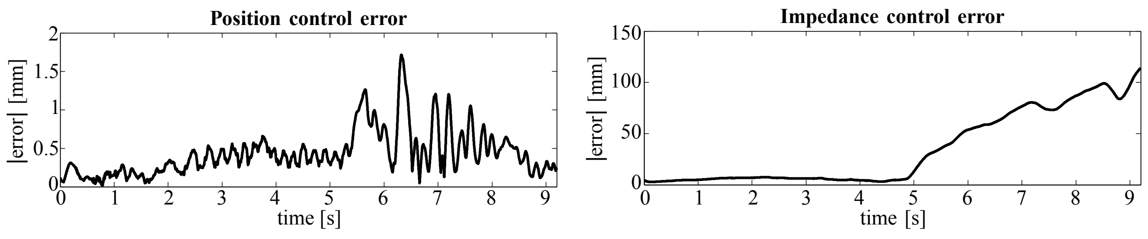


Figure 4.8.: End-effector position tracking error with position and impedance controllers. The interaction starts around 5 s in both cases. The position controller treats external forces as disturbances to reject, preventing the kinesthetic teaching. The impedance controller allows the physical guidance, but it affects the end-effector task execution.

vantages of kinesthetic teaching discussed in Section 4.1, imitation learning is one of best suited approaches for skills transfer from human to human-like robots—the so-called humanoid robots or simply humanoids—with a relatively high number of DoF. In order to reproduce the motion

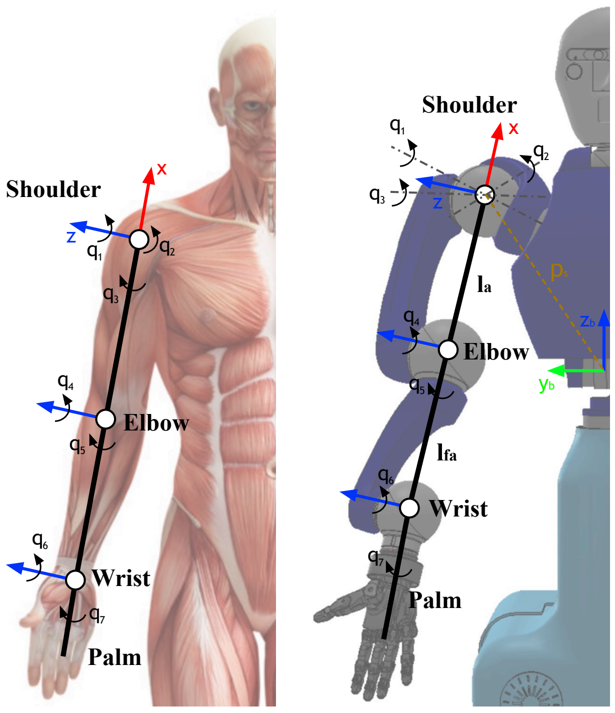


Figure 4.9.: Frames of the human and of the RoDyMan arms.

of the demonstrator on a humanoid, one has to consider that human and robot have, in general, a different kinematic structure. This problem of mapping the motion between two “entities” with different kinematics is usually referred as the motion *re-targetting* or the *correspondence* problem. The motion re-targetting problem is solved in this work using the approach presented as follows.

The problem consists in mapping the demonstrations captured with a Xsens motion capturing suit [8] from a human teacher to a humanoid robot. The Xsens measures the position and the orientation of 11 body parts: the sternum, the pelvis, and the head represent the motion of the the human trunk, while shoulders, elbows, wrists, and hands describe the motion of the two arms. The data from the Xsens are acquired at 120 Hz. The robot to teach is RoDyMan, a 21 degrees-of-freedom (DoF) humanoid robot. RoDyMan has 2 DoF in the torso, two arms with 7 DoF each, and 2 DoF in the articulated head. The end-effectors are equipped with two anthropomorphic hands in order to provide enhanced dexterous manipulation skills. The robot has also a mobile base, which is not used in this work. During the demonstration, the user can visualize the robot executing the task in a simulated environment (V-Rep [7]). This visual feedback is of fundamental importance to understand how the robot is executing the task. In order to provide the visual feedback, raw data from the suit are on-line mapped into a set of robot joint angles. The motion re-targetting problem is addressed by relying on Cartesian space variables and simple geometrical considerations. Figure 4.9 shows human and RoDyMan arms with a reference frame associated to each link. In order to transfer the motion, the position of the two end-effectors of the RoDyMan is calculated by summing three quantities, namely the position of the robot shoulder, a vector of length equal to the length of the robot arm, and a vector of length equal to the length of the robot forearm. The two vectors (arm and forearm) are oriented in the direction obtained from the Xsens arms nodes. In other words, the human hand position $\boldsymbol{x}_h$ in the RoDyMan base frame is obtained using as

$$\boldsymbol{x}_h = \boldsymbol{x}_s + \boldsymbol{R}_s \left(l_a \, \frac{\boldsymbol{x}_e - \boldsymbol{x}_s}{\|\boldsymbol{x}_e - \boldsymbol{x}_s\|} + l_{fa} \, \frac{\boldsymbol{x}_w - \boldsymbol{x}_e}{\|\boldsymbol{x}_w - \boldsymbol{x}_e\|} \right), \tag{4.7}$$

where $l_a = 0.350\,\text{m}$ and $l_{fa} = 0.305\,\text{m}$ are the RoDyMan arm and forearm lengths respectively. $\boldsymbol{s}_e$ and $\boldsymbol{x}_w$ are the positions of the elbow and wrist respectively provided by the Xsens data. $\boldsymbol{x}_s = [0, 0.1081, 0.254]^{\mathrm{T}}\,\text{m}$ is the shoulder position with respect to the base frame. In practice, the position of each end-effector of the robot is calculated from the position of the RoDyMan shoulder by summing two vectors of length l_a and l_{fa} respectively. These vectors are oriented as the relative human arm and forearm obtained from the Xsens. Notice that the positions provided by the Xsens depend on the torso movement, then it is necessary to rotate the vectors by the matrix $\boldsymbol{R}_s$ that represents the sternum orientation with respect to the Xsens world frame. For the end-effectors orientation, the hand orientations provided by the Xsens are rotated in order to maintain the coherence with the RoDyMan hand frame. Therefore, the position of the two arms are scaled taking into account the difference in link dimensions between the robot and the human.

The two computed homogeneous transformation matrices represent the target pose of the robot end-effectors and they are used in an inverse kinematics algorithm in order to control the robot (see Appendix A.2.2). Being the robot redundant, it is possible to exploit redundant DoF in order to achieve secondary goals. For the RoDyMan robot, and in general for a humanoid robot, the null-space projection of the robot Jacobian matrix can be used to guarantee that: *i)* the torso pose is bounded to prevent an excessive bending, *ii)* the shoulders maintain a natural orientation, *iii)* the elbows are confined outside of a sphere centered at the center of torso to avoid self-collisions. As detailed in the following section, this approach for motion re-targetting is effectively used in this work to transfer structured tasks form a human teacher to the RoDyMan robot.

4.3. An integrated framework for learning structured tasks

The intuitive transfer of structured tasks from a human demonstrator to a (humanoid) robot requires the following steps:

- A human teacher demonstrates the structured task. A low-lever control layer is required to permit the kinesthetic teaching or to perform motion re-targetting.
- The provided demonstration(s) is segmented into basic motion units and a unique label is assigned to each basic motion. This requires a reliable segmentation strategy.
- Segmented data are used to create a database of motion primitives. Each motion primitive is represented using one of the solutions presented in Chapter 3. Learned motion primitives are used at run time to generate the robot's trajectory and effectively execute the task.
- Generated action labels, together with execution constraints, are automatically associated to the task structure. To this end, an high-level process has to monitor the demonstration phase and decide where the last segmented action has to be attached. The learned structure is exploited to generate the task plan, as discussed in Section 5.2.

The intuitive transfer of structured tasks is realized using the framework illustrated in Figure 4.10. The framework integrates two main components, namely a *Robot Manager* and an *Attentional Supervisory System*. The *Robot Manager* (RM) handles low-level aspects of the transfer process, like task supervision, execution, and learning. In particular, the RM permits the skills transfer via kinesthetic teaching or human imitation, it is responsible for on-line segmentation, motion primitives learning and execution. In the considered scenario, the human can interact with

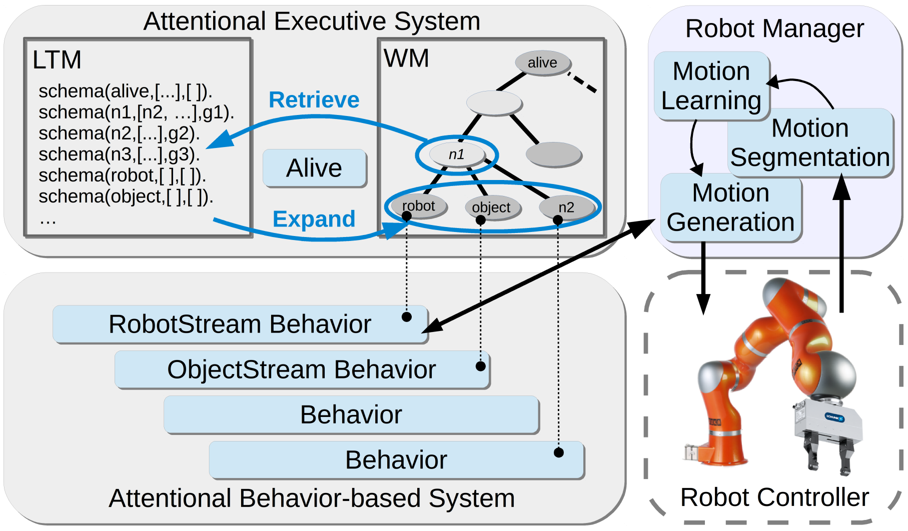


Figure 4.10.: The overall framework for intuitive transfer of structured tasks. The attentional system supervises task execution and learning, while the *Robot Manager* enables the segmentation of the robot activities (*Motion Segmentation*), the kinesthetic teaching or motion re-targetting, the motion primitives learning (*Motion Learning*) and execution (*Motion Generation*). The attentional system manages the execution of high-level tasks (*Attentional Executive System*) and low-level sensorimotor processes (*Attentional Behavior based System*). The communication between the *Robot Manager* and the attentional system is managed by the *RobotStream* (robot motion data) and *ObjectStream* (perceived data from the RM to the attentional system).

the robot in a multimodal manner with speech and physical guidance during the teaching session. An attentional system supervises both the human and the robot activities (*Attentional Behavior based System*), and it handles high-level aspects of the transfer process, like task monitoring and execution (*Attentional Executive System*). Further details are given as follows.

4.3.1. Robot Manager

The Robot Manager (RM) handles low-level aspects of the human-robot interaction and it is responsible for a correct task execution. In particular, the RM is responsible for: *i)* smooth transition between teaching and execution modes; *ii)* segmentation of the human demonstration into basic actions; *iii)* scene monitoring (objects classification and tracking); and *iv)* robot state monitoring (robot–object distances, motion primitives learned or executed). Task teaching is performed by means of kinesthetic teaching for a single robotic manipulator (see Section 4.1) or by imitating human motions for a humanoid robot (see Section 4.2). Kinesthetic teaching is allowed by the gravity compensation control in Appendix A.2.3, which makes the robot ideally weightless for an easy and safe physical guidance. Imitation learning, instead, is performed by applying the kinematic controller described in Section 4.2. Structured tasks are segmented into a set of point-to-point motions (reaching and manipulating objects). Segmented data are compactly represented as stable dynamical systems (DS), called in this work the motion primitives. As discussed in Chapter 3, stable DS are well-suited for point-to-point motion generation since they are guaranteed to converge towards a given target. The learned motion primitives are used at run time to generate the robot's trajectories and execute the structured task.

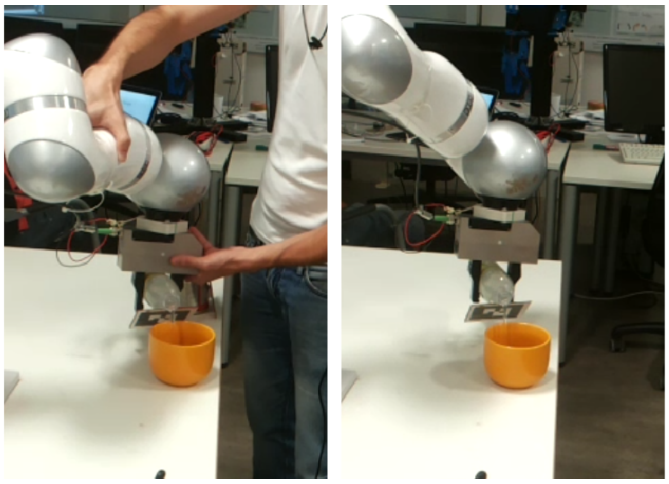


Figure 4.11.: Teaching and execution of the *pouring* action. (Left) During the teaching the user drives the robot near the cup and pours water. (Right) During the the execution the robot reproduces the learned skill.

Action segmentation

The demonstrated task has to be segmented into elementary movements. As discussed in [53], an effective segmentation strategy has to be fast enough to work in real-time, consistent across different demonstrations of the same task, and complete, meaning that the set of all generated segments represents the entire task. In this work, a simple and effective segmentation mechanism is exploited. The approach is based on object proximity and explicit human commands. Following the approach in [156], each object in the environment is associated with a proximity area, namely a sphere of radius r around each object. When the end-effector of the robot enters or leaves the proximity area of an object, a new segment is generated. Analogously, when a human command (open or close the robot's hand) is executed a new action is created. The attentional system can then automatically connect the generated action segments to the task structure (see Section 4.3.2), while the RM uses the robot's trajectories to learn a motion primitive for each action segment. Human commands are also included in the task structure, in order to control the gripper when the robot executes the task. Two classes of actions are considered:

- Near Object Action (NOA): the action is segmented inside the proximity area of an object. This class include complex movements like pouring or mixing, which are accurately and compactly encoded into dynamic movement primitives (DMPs). As discussed in Section 3.1.1, DMPs are learned from a single demonstration. Learned DMPs are used to accurately reproduce the motion on real robots, as shown in Chapter 5.

- Far Object Action (FOA): the action is segmented outside the proximity area of an object. In this case, only the end-point of the observed trajectory is considered. The action is then reproduced with a point-to-point motion, generated with a linear dynamical system. In this way, the robot reaches the proximity area always with the same pose, and executes the NOA starting from a state which is consistent with the demonstration. This is useful, for example, to grasp an object, since the robot reaches the same pre-grasping pose and then grasps the object always in the same manner.

The described segmentation mechanism allows the system to reproduce complex actions involving two or more objects. For example, the pouring action (NOA) illustrated in Figure 4.11 has been trained with high accuracy and associated with the *pour(water)* primitive behavior within the abstract task of pouring a drink.

It is worth noticing that, since the segmentation strategy requires the robot–object distances, possible failures may occur if the objects are not properly tracked. For instance, this is the case if the teacher hides the object to manipulate during the teaching. Failures may also occur if the robot enters in the proximity area of multiple objects simultaneously and each of these can be associated with the generated segment. This occurrence can be prevented by properly choosing the radius of the proximity area r. Finally, the segmentation strategy may generate unnecessary segments if the teacher guides the robot inside/outside the proximity area of different objects without grasping them. Even in this case, the learned task can be correctly executed although the robot performs unnecessary motions like the human demonstrator. This undesirable behavior can be prevented by instructing the teacher to directly guide the robot towards the object to use.

Learning motion primitives

The described segmentation approach generates a set of point-to-point motions with associated target poses. Segmented data are represented as stable dynamical systems, called motion primitives in this work. Motion primitives are learned from a single demonstration using the dynamic movement primitives (DMPs) approach presented in Section 3.1.1. DMPs, in fact, are well-suited to represent stable motions given a single demonstration. Note that, in this work, DMPs are used to represent both the position and the orientation of the end-effector, and that the orientation is represented using the roll-pitch-yaw Euler angles [142].

The full DMP structure described by equations (3.3a)–(3.3c) is exploited to learn and retrieve Near-Object-Actions, which correspond to the complex movements that the robot has to perform inside the proximity area. For instance, the motion in Figure 4.11 is a NOA. Recall that the clock signal in Equation (3.3c) depends on the tunable parameter $\gamma > 0$, and that, in practice, $s = 0$ after $5/\gamma$ seconds. Hence, one can simply set the parameter γ of each motion primitive equal to 5 divided by the time duration of the relative NOA. It is worth noticing that the goal pose of each NOA (DMP) is automatically retrieved from the segmented data, i.e. the last point of the trajectory is considered as the goal of the DMP.

Far-Object-Actions are represented by linear dynamical systems. In oder words, the non-linear terms $-\boldsymbol{K}(\boldsymbol{g}-\boldsymbol{p}_0)s+\boldsymbol{K}\boldsymbol{f}(s)$ in Equation (3.3b), as well as the clock signal in Equation (3.3c) are simply neglected. Also for FOA, the goal pose is automatically retrieved from the segmented data. In this way, the robot follows a linear trajectory far from the objects, and it executes complex actions always from the same initial state. This is useful to grasp objects always from the same relative pose, and to prevent the problem of the excessive magnification of trajectories generated from different initial states [65].

It is useful to point out once more that the combination of DMPs and the segmentation strategy permits to learn motion primitives without any off-line data processing. In particular, the target pose for each action, as well as the demonstrated trajectory, are automatically provided by the segmentation approach and then used to learn the motion primitives, without further human intervention or data post-processing.

4.3.2. Attentional System

The attentional system provides the cognitive control mechanisms needed to flexibly orchestrate the execution of structured tasks and to monitor the human activities. The framework presented in this work follows a Supervisory Attentional System (SAS) approach [40, 110], where interactive action execution and learning are supported by attentional mechanisms. In a SAS framework,

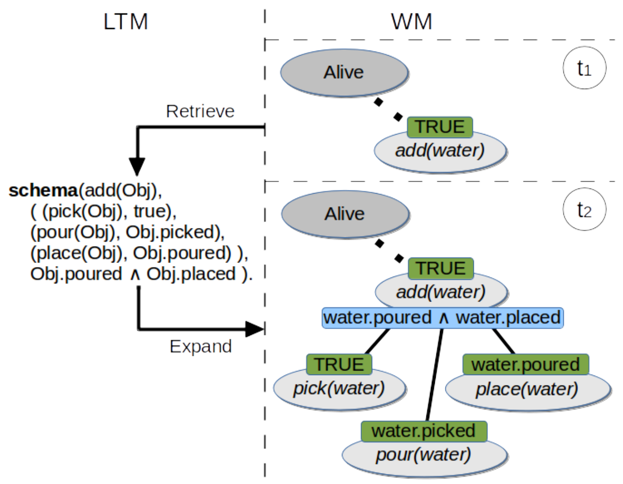


Figure 4.12.: Representation of the WM expansion process managed by *alive*. When the new node *add(water)* is allocated in WM the associated schema is selected from LTM (retrieve phase) and exploited to decompose the node in WM by adding and instantiating the new abstract or concrete sub-nodes mentioned in the schema (expand phase).

the executive control depends on two main mechanisms: contention scheduling and supervisory attention. The first one allows to reactively select and regulate routinized activities depending on bottom-up perceptual stimuli and internal drives. The latter is a higher-level process that drives the system towards task-oriented behaviors via attentional regulations. In a human–robot interaction scenario, the attentional system exploits a hierarchical task representation (the rooted tree presented in Section 3.3) to supervise and regulate the robot actions, while interacting with the human. As proposed in [26, 27], the framework consists of a Long Term Memory (LTM) and a Working Memory (WM) (see the Attentional Executive System in Figure 4.10). Note that, as detailed below, both the LTM and the WM exploit the hierarchical representation of structured tasks presented in Section 3.3.

Long Term Memory

The Long Term Memory (LTM) contains the procedural knowledge available to the system, or, in other words, the abstract description of the tasks that the robotic device can execute. Equation (4.10) presents an instance of the abstract task description. More specifically, each task is hierarchically defined in the LTM by a set of predicates of the form **schema**(m, l, p), where m is the name of the task, l is a list of m_i subtasks associated with enabling conditions r_i (releasers), i.e. $l = \langle(m_1, r_1), \dots, (m_n, r_n)\rangle$, while p represents a post-condition used to check the accomplishment of the task. These task definitions are exploited to be retrieved, allocated, and instantiated in the WM for execution, as illustrated in Figure 4.12. To this end, a special process named *alive* is continuously called to update the WM by allocating and deallocating a hierarchy of behaviors that implements the corresponding task schemata in the LTM. For instance, in Figure 4.12, the $add(Obj)$ schema is retrieved by *alive* (t_1 step in Figure 4.12) and then instantiated and allocated in the WM as the behavior $add(water)$, which is ready for the execution (t_2 step in Figure 4.12).

Working Memory

The Working Memory (WM) is used to temporarily maintain and manipulate the information needed to execute task-oriented activities. The WM constitutes the executive state of the attentional system and collects the processes recruited and instantiated for task execution. In this framework, these processes are represented by an annotated rooted tree $T = (r, \mathcal{B}, \mathcal{E})$, where the nodes $\mathcal{B}$ represent allocated processes/behaviors, the root $r \in \mathcal{B}$ is the *alive* process, which bootstraps and manages the WM, while the edges $\mathcal{E}$ represent parental relations among sub-processes/sub-behaviors. The nodes $\mathcal{B}$ are *concrete* if they represent real sensorimotor processes or atomic actions (see Section 3.3). The nodes are *abstract* if they represent complex behaviors that can be hierarchically decomposed. Figure 4.12 (bottom, right) illustrates an example of the hierarchical behaviors generated for the execution of the *add(water)* subtask. The behaviors generated for the pouring task are shown in Figure 4.13. In both figures, green labels represent *releasers* (enabling-conditions), while blue labels are post-conditions (*goal* conditions) exploited to check the accomplishment of goal-oriented activities. As discussed in Section 3.3, each node $\mathcal{B}$ in the WM consists of a 6-tuple $(m_b, r_b, p_b, x_b, e_b, \mu_b)$, where m_b is the name of the allocated task, r_b and p_b represent the releaser and post-condition respectively, x_b is the set of sub-behaviors generated by m_b, e_b is an emphasis value, and μ_b is a magnitude value. Here, m_b, r_b, p_b, x_b are instances of the associated schema in the LTM. Indeed, each node $\mathcal{B}$ in the WM is generated by the *alive* process that allocates a $\mathbf{schema}(m, l, p)$ with a variable binding that instantiates m in m_b, p in p_b, and the list l in the list of sub-behaviors x_b and the associated releasers. As an example, the $pour(Obj)$ task in Figure 4.13 is instantiated by the argument *water* for the variable *Obj*. Hence task, releaser, post-condition, and sub-behaviors are also instantiated by *water*. This process is analogous to the one introduced in [104] for HTN planning. Each concrete behavior accesses sensory data σ_b, affects control variables c_b, and updates a set of state variables V representing the current state of the overall system. For example, when the *water.picked* boolean variable in Figure 4.13 is set to true, the *pick* task is accomplished while the next *pour* task is enabled to be executed. In this case, the state variable *water.picked* is updated by the concrete behavior *gripper(close)* when the grasped object is the water.

Attentional regulation

As discussed in Section 3.3, and in line with with [110], each node in the WM is also endowed with an activation value regulated by attentional mechanisms. This value is affected by top-down and bottom-up attentional processes. In the presented framework, concrete behaviors are periodically monitored and controlled. The frequency of monitoring sensorimotor processes is the activation value of concrete behaviors. More specifically, in concrete behaviors, the activation value is bottom-up regulated by a monitoring function $g(\sigma_b, \varepsilon_b) = \lambda_b$, which depends on behavior-specific stimuli σ_b and behavioral state variables ε_b. Note that the variables ε_b are a subset of the state variables V. As proposed in [27], the distance of the target is used as an estimation of the behavioral accessibility and σ_b is directly associated to the minimum distance of the target for the behavior. In particular, the activation period $\lambda_b \in [\lambda^{min}, \lambda^{max}]$ is bottom-up regulated by

$$\lambda_b = g(\sigma_b, \varepsilon_b) = \begin{cases} \lambda^{min} & \text{if } \sigma_b \leq r^{min} \\ \lambda^{max} & \text{if } \sigma_b \geq r^{max} \\ \alpha \cdot \sigma_b + \beta & \text{otherwise} \end{cases}, \tag{4.8}$$

where the two parameters r^{min} and r^{max} are the minimum and maximum values of λ respectively. The function $g(\cdot)$ linearly grows from λ^{min} to λ^{max}. The parameters $\alpha = (\lambda^{max} - \lambda^{min})/(r^{max} - r^{min})$ and $\beta = \lambda^{min} - \alpha \cdot r^{min}$ describe the linear increase of g for σ_b in the interval $[r^{min}, r^{max}]$. Notice that the activation period in Equation (4.8) is only influenced by the σ_b stimuli, or, in other words, that ε_b is not exploited in this work. However, more complex regulations can be introduced (see, for example, [24]).

The bottom-up regulation in Equation (4.8) is then top-down modulated by a magnitude value μ_b that summarizes the overall influence of the WM on the behavioral attentional state. In concrete behaviors, top-down and bottom-up influences are combined in the *emphasis* value $e_b = \mu_b/\lambda_b$, which represents the actual activation frequency for the behavior b. The absence of a top-down influence is represented by $\mu_b = 1$. Whenever a magnitude change happens for a node in the WM, this update is inherited by all its descendants. In addition, when a behavior is accomplished, the magnitude of the parent node is increased by a constant value k, which is then propagated towards its active successors. This implies that the magnitude of a generic behavior is given by $\mu_b = \mu_f + kn$, where μ_f is the parent magnitude and n is the number of accomplished sub-behaviors. This mechanism facilitates active behaviors representing the continuation of accomplished subtasks and reduces the number of task switches. For example, after executing the *pick(water)* action in Figure 4.12, the subtask *add(water)* becomes more relevant and the system will choose the *pour(water)* action also in case of conflicts between subtasks.

Conflict regulation

Structured tasks consists of a several subtasks (see Section 3.3 and Figure 4.14). Some of the subtasks have to be executed with a fixed order, some can be arbitrary accomplished. For example, considering the tree structure in Figure 4.14, *use(spoon)* has to be executed after *add(coffee)* and *add(water)* (releasers *water.added* and *coffee.added*). On the contrary, subtasks *add(coffee)* and *add(water)* have both a *true* releasers and can be arbitrary accomplished. The selection of the next subtask to execute is treated as a conflict regulation problem, and the behavioral activation level is exploited to regulate behavioral competitions and conflicts. Indeed, multiple subtasks can be allocated in the WM at the same time (for instance *add(coffee)* and *add(water)*), therefore several behaviors can compete for the execution generating conflicts and impasses [20]. Contentions among alternative concrete behaviors are solved exploiting the attentional activation: following a winner-takes-all approach, the behaviors associated with the higher emphasis are selected with the exclusive access to mutually exclusive resources. More specifically, in the attentional system this mechanism occurs when multiple concrete behaviors simultaneously try to access and update a mutually exclusive control variable c. In this case, given the set $C(c)$ of subtasks that compete for c, the system selects the most emphasized concrete behavior

$$b_{win} = \operatorname*{argmax}_{b \in C(c)} e_b. \tag{4.9}$$

The selected behavior b_{win} can then modify the variable c with the exclusive access. As previously described, once a behavior is accomplished, the upward propagation of magnitude facilitates task-related behaviors in case of conflicts and orients the system towards task continuation and accomplishment. This task-oriented facilitation mechanism can be enhanced or reduced by tuning the parameter k.

4.3.3. Teaching uni-manual structured tasks

The presented framework supports human–robot interaction during both task demonstration and task execution. In order to enable natural interaction and incremental task learning, the system can anytime switch between teaching and execution. The teaching phase can start from the human or the robot initiative. In the first case, the human commands to switch to a demonstration session and

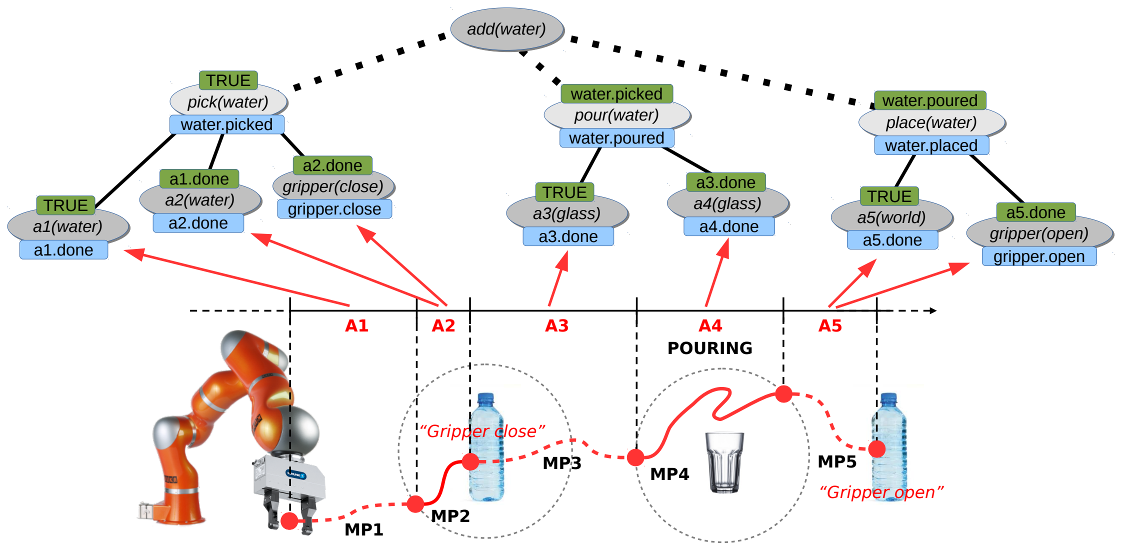


Figure 4.13.: Action segmentation and hierarchical task decomposition during the kinesthetic teaching of a pouring task. The robot has to pick-up the bottle (*pick(water)*), reach the glass, pour the water (*pour(water)*), and place the bottle (*place(water)*). The Robot Manager (down) performs action segmentation (A_1, A_2, …, A_5) and learns the associated motion primitives (MP_1, MP_2, …, MP_5), while the attentional system (up) connects the generated action segments to the task structure ($a1(water)$, $a2(water)$, and $gripper(close)$ connected to $pick(water)$; $a3(glass)$ and $a4(glass)$ connected to $pour(water)$, etc.). The green and blue labels represent releasers and post-conditions respectively.

directly show the execution of a task. In the latter case, the robot waits for the human assistance when it is not able to accomplish a task. This happens when a task under execution is not linked to concrete sensorimotor behaviors. In this case, the system waits for the user guidance in order to learn how to perform the missing subtasks.

During the teaching phase, the human physically guides the robot in order to demonstrate the correct execution of the task. This kinesthetic teaching session is supervised by the attentional system, which has to connect the segmented atomic actions to the related subtasks. The attentional system exploits an initial (or empty) task structure that is assumed to be given, for example by an expert user. An example of the initial task structure is shown in Figure 4.14. Depending on the task at hand, the initial structure can have two or three levels, and it is completely specified up to the subtasks level (see Section 3.3). The attentional system tracks and monitors both the human and the robot task execution, and it attaches an incoming segmented action to the most emphasized subtask, learning a task representation that can be effectively used to execute the task.

The task learning process is illustrated in Figure 4.13. In particular, the figure shows the action segmentation of a pouring task along with the associated hierarchical task decomposition. During the kinesthetic teaching, the RM handles action segmentation and motion primitive learning, while the attentional system monitors the subtasks to be fulfilled (i.e. *pick(water)*, *pour(water)*, and *place(water)*). The RM segments the demonstrations into basic actions and associates a unique label to each action. These labels are indicated with $A_1, A_2, \ldots, A_5$ in Figure 4.13. The RM sends the action labels to the attentional system, which creates new nodes to the tree ($a_1(water)$, $a_2(water)$, $\ldots$, $a_5(word)$) and links each node to the most emphasized subtask. During the demonstration, the human directly controls the robot's gripper with a verbal command. In order to reproduce the

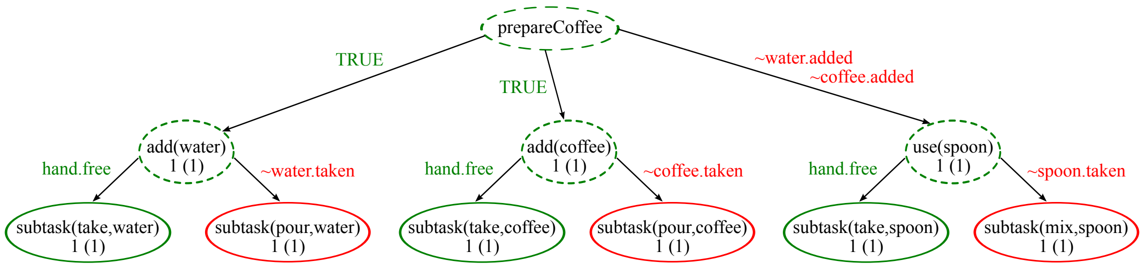


Figure 4.14.: The initial task structure for the task of prepare a coffee. During the teaching phase, the attentional systems attaches the atomic actions to this initial task structured.

task, the actions *gripper(close)* and *gripper(open)* are generated after a verbal cue and attached to the task structure. Moreover, the human can always inspect the result of a training session by invoking the repetition of learned tasks and subtasks. A novel task demonstration can be provided at any time if the learned activities are not satisfactory.

The process of attaching the generated action labels to the task structure is managed by the attentional system while monitoring the human demonstration. The procedure assumes that a partial description of the task structure is already provided in the LTM (see Equation (4.10) and Figure 4.14 for an example). Specifically, the pre-loaded hierarchical structure is composed by an abstract task node, eventual concrete tasks, and a set of dynamic *subtasks* (see section 4.3.2), everyone associated with an object in the scene (see Figure 4.15, t1). The goal of the learning process is then to produce an updated LTM where all the associations between subtasks and actions are represented. In this work, *open subtasks* are the schemata of LTM that represent concrete behaviors (as *subtask(take,Obj)* in Equation (4.10)) but are not associated to atomic actions (i.e. that cannot be executed by the robot). Starting from an initial LTM^0 that contains a set of n open subtasks, the learning process produces an updated LTM^l where the open subtasks in LTM^0 are further decomposed by the m generated actions, each associated with a motion primitive in the Robot Manager. When a teaching phase starts, the abstract behavior representing the task to be demonstrated is allocated in the WM and then hierarchically decomposed by the *alive* process (see Section 4.3.2). In this way, a behavioral tree T_{task} is generated in the WM that contains a set of open subtasks $O = \{subtask_1, \dots, subtask_n\}$ to be linked to the action labels produced by the RM.

The learning process is shown in Figure 4.15 for a water-pouring task. This task is hierarchically decomposed in the *take-water* and *pour-water* subtasks (frame $t1$), which are denoted in the LTM by the following schemata:

$$\begin{aligned}
&\textbf{schema}(add(Obj),\\
&\qquad \langle(subtask(take,Obj),hand.free),\\
&\qquad (subtask(pour,Obj),Obj.taken)\rangle,\\
&\qquad Obj.used)\\
&\textbf{schema}(subtask(take,Obj),\langle\ \rangle,Obj.taken)\\
&\textbf{schema}(subtask(pour,Obj),\langle\ \rangle,Obj.used)
\end{aligned} \tag{4.10}$$

Notice that the *take* and *pour* subtasks can be instantiated with different objects (*Obj*). The subtask *take* is enabled when the hand is free (releaser *hand.free*) and associated with the post-condition *Obj.taken*, while the *pour* subtask is enabled when the object is taken (releaser *Obj.taken*) and related to the *Obj.used* post-condition.

In order to be executed, the *add(Obj)* has to be instantiated and allocated in the WM. However,

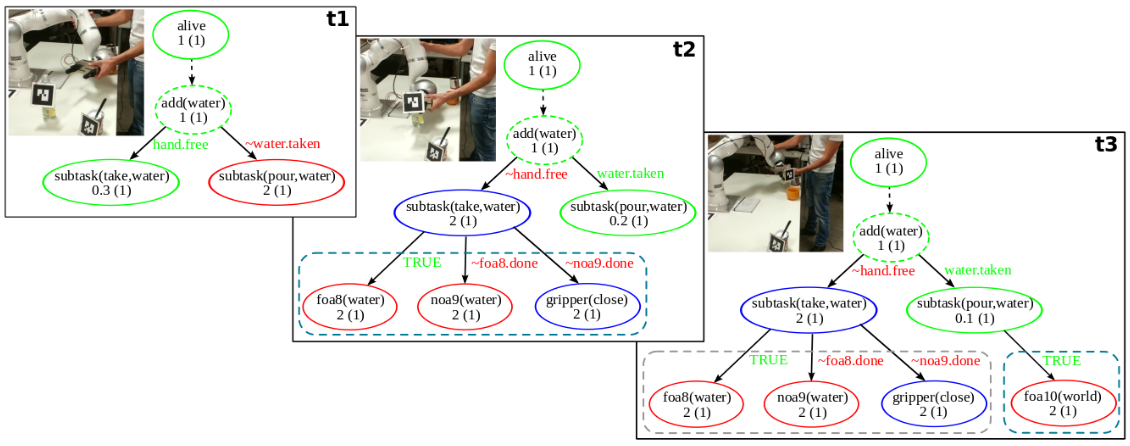


Figure 4.15.: Representation of the WM update during the demonstration of the *pouring* task. The system starts from a simple structure for the *add(water)* task (t1). During the demonstration new actions are added to the *take-water* subtask (t2) along with their releaser (labels on the arrows). When the new *pour-water* subtask is selected (t3) a new FOA is linked with a *true* releaser. Here, green and red ovals represent enabled and disabled behaviors (satisfied and unsatisfied releasers), blue ovals are for accomplished behaviors (satisfied postconditions), dotted ovals are for abstract behaviors. For each behavioral node, the values outside/inside brackets are for the inverse of emphasis $1/e_b$ (i.e. activation period) and magnitude μ_b (top-down influence) respectively.

the two subtasks *pour* and *take* are not linked to concrete sensorimotor processes, which are automatically generated during the kinesthetic teaching. Since each subtask is implemented by a concrete WM node, it is associated with an activation level, that is bottom-up affected by the proximity of the objects in the scene (see Equation (4.8)) and top-down modulated by the overall tasks allocated and enabled in the WM. Therefore, during the demonstration, the attentional system enhances the activation of the subtasks which are closer to the associated target objects (accessible) and top-down stimulated through the task structure (task relevant). These activation values are then used to link the concrete subtasks to the generated actions, as summarized in Algorithm 4.

In particular, when a new atomic action is generated by the Robot Manager (line 2), all the enabled open subtasks in the WM compete to add the action as a new child node (line 3). This competition is managed by a winner-takes-all approach where the most emphasized subtask acquires the new action (see Equation (4.9)). In order to add a novel action node, one has to define its releaser and post-condition (lines 4-9). The releaser is always enabled (*true*) if a FOA is added to a subtask with no other child nodes (lines 4,5). Otherwise, the execution of the actions has to be chained, hence the post-conditions of the previous action is employed as the releaser of the current one (lines 6,7). The post-condition of each action is then set to *a.done* (line 9). This symbolic post-condition is associated with a sub-symbolic constraint used to check whether the associated motion has been actually executed by the robot. If an action is associated with a motion primitive, sub-symbolic constraints are directly provided by the RM (e.g. if the end-effector reached the target). Instead, predefined commands like open and close the gripper are directly associated with predefined sub-symbolic conditions, as constraints on the gripper state. When a new action is generated, a corresponding new behavior is allocated in WM as a child node of the winning open subtask (line 10) and the LTM is updated accordingly (lines 11,12). The WM update is also shown in Figure 4.15, frame t2, where the linked actions are indicated by the dotted line. Notice that the chaining constraint is introduced for actions belonging to the same subtask, or if the subtask

Algorithm 4 Allocation of a new action in the task hierarchy.

```
1: while true do
2:     if a new action $a$ from RM exists then
3:         get winner $sub_{win} \leftarrow \operatorname{argmax}_{sub_i \in O} e_{sub_i}$
4:         if $sub_{win}$ is a new subtask and $a$ is FOA then
5:             set releaser $q \leftarrow true$
6:         else
7:             set releaser $q \leftarrow p_{prev}$
8:         end if
9:         set post-condition $p \leftarrow a.done$
10:        add behavior $(a, q, p, \emptyset, \mu)$ to $sub_{win}$ in WM
11:        add new schema$(a, \langle\ \rangle, p)$ in LTM
12:        add $\langle a, q \rangle$ to subtask list of $sub_{win}$ in LTM
13:    end if
14: end while
```

starts with a NOA, which requires the fixed starting point provided by the previous action. On the other hand, a new subtask starting with a FOA is kept decoupled from the previous subtask, allowing re-usability and flexible execution of the associated subtask. Indeed, during the execution, all the enabled subtasks of the WM compete to acquire the control of the robotic platform. This implies that multiple independent subtasks can be executed in a flexible manner, diverging from the demonstrated sequence. The overall structured task acquisition process is exemplified in Figure 4.15. Once the user drives the robot towards the bottle and grasps it, the system generates 3 new actions: *foa8(water)* when the robot enters the proximity area of the bottle, *noa9(water)* when the bottle is reached, and *gripper(close)* when the bottle is grasped. These actions are attached to the *take-water* subtask, which is the only enabled subtask. The enabling conditions associated to each action are needed to ensure that *noa9* is executed after *foa8*, and *gripper(close)* after *noa9*. Afterwards, when the robot is driven towards the cup, the novel action *foa10(world)* is generated and linked to the *pour-water* subtask, activated after that the bottle is grasped. In this case, the motion between the bottle and the cup represents a FOA which is associated to the subtask with a *true* releaser.

The results of the learning process, for the task of preparing a coffee, are shown in Figure 4.16. The learned structured task consists of 27 atomic actions, attached to the 6 subtasks already defined in the initial structure (see Figure 4.14). Each action also contains releaser and post-conditions. In particular, the first action of each subtask has always a *true* releaser. A generic action in a subtask can be activated only if the previous one has been successfully completed. For example, *noa9(coffee)* is executed only if *foa8* is terminated. The tree structure in Figure 4.16 also contains gripper specific actions, i.e. *gripper(open)* and *gripper(close)*. Those actions are generated after an explicit speech command. The learned task structure, together with the associated motion primitives, allows the execution of the structured task on real robots, as detailed in Section 5.2.

4.3.4. Teaching dual-arm structured tasks

In this work, a dual-arm (or multi-arm) task indicates any task executed by robots equipped with two (or more) arms, without considering the problem of generating and executing synchronized arm trajectories. Hence, dual-arm differs from bi-manual, since a bi-manual task requires the synchronization of the generated motion trajectories.

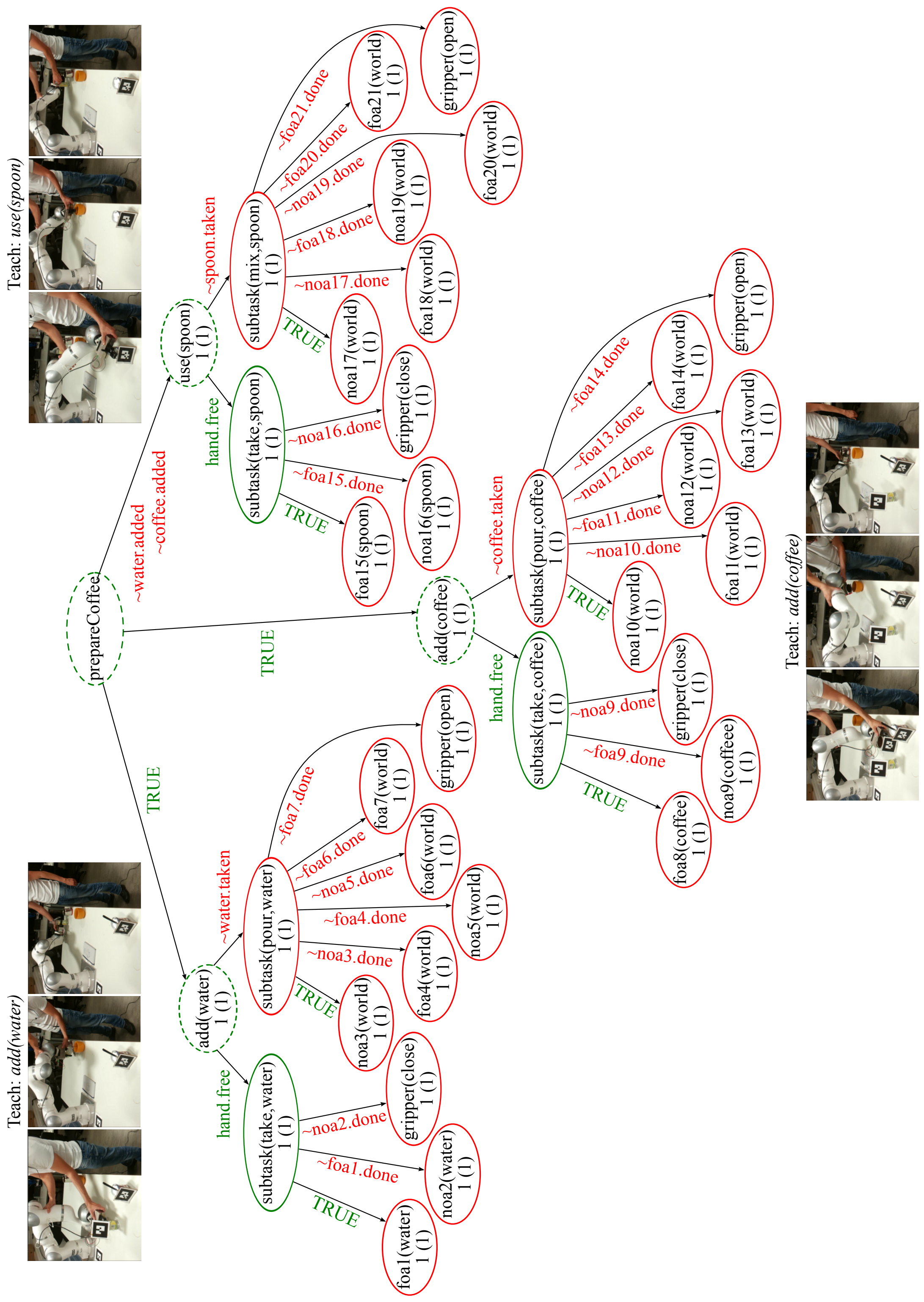


Figure 4.16.: The tree learned after a kinesthetic demonstration of the task of prepare a coffee. Concrete tasks and subtasks are assumed to be know, while the atomic actions with associated pre- and post-conditions are learned from the task demonstration.

Given this definition of dual-arm tasks, the approach presented in the previous section to teach uni-manual structured tasks can be easily extended to consider dual-arm tasks. In particular, each arm is controlled separately by considering a separate Robot Manager (see Section 4.3.1) for each arm, namely the left and right arm managers. The task structure is also duplicated to have a different structure for each arm, and the novel attribute *arm* is added to each node in the structure. The attribute is a unique label that identifies the relative arm. As for the uni-manual case, the actions segmented during the task demonstration have to be connected to the initial task structure and stored in the system repository (LTM). The difference is that, in the dual-arm case, two task structures are considered (on for the left and one for the right arm). Moreover, differently from the uni-manual case, here the human motion is mapped to the humanoid robot by using the approach described in Section 4.2, instead of using kinesthetic teaching.

For instance, considering a simple pouring task, the activity can be hierarchically decomposed in the *take* and *pour* subtasks, which are denoted in the LTM by the following schemata:

$$\begin{aligned}
&\mathbf{schema}(add(A,O),\\
&\qquad\langle(subtask(A,take,O),A.hand.free),\\
&\qquad(subtask(A,pour,O),A.O.taken)\rangle,\\
&\qquad O.used).\\
&\mathbf{schema}(subtask(A,take,O),\langle\,\rangle,A.O.taken).\\
&\mathbf{schema}(subtask(A,pour,O),\langle\,\rangle,O.used).
\end{aligned} \tag{4.11}$$

The variable O in (4.11) represents the target of the pouring, while A is the arm involved in the action execution (*left* or *right*). Note that the attribute A is the only difference between the schemata in (4.11) used to decompose dual-arm tasks, and the schemata in (4.10) used to decompose uni-manual tasks. The schemata in (4.11) also contains pre- and post-conditions: the *take* subtask is enabled when the hand of the arm A is free (A*.hand.free*), while the *pour* subtask is enabled when the left (right) hand holds the object ($A.O$*.taken*). Moreover, the latter subtask is finalized when the target-object O is used. Notice that, if both arms are enabled to execute a task, two instances of the related schema are allocated in WM with different values for A. Being these instances in conflict, only one arm will execute the task. During the human demonstration, the attentional system continuously monitors the environment and the task structure exploiting top-down and bottom-up regulations to enhance the activation of accessible and task relevant subtasks. When the two arm managers recognize a new action all the left/right labeled subtasks compete to acquire the related action. As for the uni-manual case, the activation values are used to manage the competition following a winner-takes-all approach, where the most activated subtask acquires the action. Pre- and post-conditions are determined as in the uni-manual case, but for each arm separately.

4.4. Summary and conclusion

This chapter presented approaches for intuitive skills transfer from a human teacher to a (humanoid) robot. The first section discussed the problem of intuitively transfer uni-manual skills, or, in other words, motions executed by a single robotic manipulator. Kinesthetic teaching, i.e. physically guide the robot in order to demonstrate the task, is the most effective technique to transfer uni-manual skills. An effective approach has been presented to teach end-effector and null-space motion primitives in a unified manner. The approach exploits incremental learning techniques to iteratively refine previously acquired skills, and a customized multi-priority kinematic controller to permit a safe human–robot interaction during the motion reproduction. Experiments on a real robot with seven degrees-of-freedom have shown the effectiveness of the presented approach, which effectively enables a user to intuitively teach novel skills. A limitation of this approach is that it considers variable task priorities during the kinesthetic teaching, but fixed priorities during the execution. This means that the user has to predefine the priority of end-effector and null-space motions. An interesting research work consists in overcoming this limitation by considering variable task priorities during the execution. A possible way to find the task priorities can be solving a constrained optimization problem, where the cost function minimizes the end-effector and null-space execution errors. The constraints are used, for example, to take into account robot's physical limitations, or to guarantee that the demonstrated final configuration of the robot is reached.

In case of robots with many degrees-of-freedom, as humanoid robots, kinesthetic teaching is not a suitable option, because the teacher has to handle several degrees-of-freedom at the same time. A better approach, instead, consists in imitating human movements tracked with a dedicated system like the motion capturing suit used in this work. Section 4.2 presented an approach for motion mapping between a human demonstrator and a humanoid robot. Human and humanoids have, in general, a different kinematic structure with different joint limitations, which makes the motion re-targetting problem not trivial. The approach presented in Section 4.2 exploits simple kinematic considerations to solve the motion re-targetting problem. The approach is simple to implement and works in real-time with a relatively high number (up to 21) of degrees-of-freedom. The approach is used to control the RoDyMan humanoid robot, which has a human-like torso but a mobile base instead of the legs. This greatly simplifies the motion re-targetting problem since the robot's balance does not have to be considered. This is not the case when the motion has to be transferred to legged robots. For legged robots one has to prevent possible falls during the task execution by ensuring the robot balancing with a customized controller. Approaches have been proposed to solve the balancing problem (see, for example, the work in [63, 64]) and can be easily integrated in the presented framework.

The last section focuses on intuitive transfer of uni-manual and dual-arm structured tasks. The intuitive transfer of structured tasks is realized by combining the skills transfer approaches presented in Section 4.1 and 4.2 with an attentional system. In particular, a human demonstration is firstly segmented into the basic motion units constituting a structured task. Segmented data are then used learn a set of motion primitives represented as dynamical systems (see Section 3.1). Moreover, the attentional system continuously monitors the task demonstration and exploits a mechanism of cognitive control to connect the segmented actions and associated pre- and post-conditions to a partially specified task structure. The same framework for intuitive transfer of structured tasks is exploited in Chapter 5 to execute the learned task autonomously or in cooperation with a human co-worker. One limitation of the presented framework is that the task structure is partially specified by an expert user until the so-called subtasks level, while the leaves level is learned from a task demonstration. An interesting research direction consists in learning the entire tree structure from human demonstration(s). To this end, it is possible to exploit hierarchical clustering algorithms to create and connect the nodes of the tree, while the associated logical conditions can be potentially inferred using context-free grammars.

This chapter, and in particular Section 4.3.4, proposed a solution to teach dual-arm tasks, introducing a clear difference between dual-arm and bi-manual tasks. In a dual-arm task, the problem of generating synchronized trajectories is not considered, which simplifies the learning process. On the contrary, bi-manual tasks require a synchronized generation and execution of the motion primitives. An interesting research direction is to extend the framework for intuitive teaching to structured bi-manual tasks. To this end, several problems have to be solved. First, the skill representations presented in Chapter 3 are not suitable for bi-manual task representation, since they do not consider the synchronization between left and right arm movements. The work in [54,143,152] are state-of-the-art representations of bi-manual skills. In particular, the work in [54, 152] extend the standard dynamic movement primitives in order to generate synchronized motions, and they can be more easily integrated in the presented framework compared to [143]. The second problem to solve is the automatic segmentation and recognition of bi-manual actions. To this end, it is possible to extend the segmentation strategy presented in Section 4.3.1 to the bi-manual case, for example by considering an action bi-manual when both hands are in the proximity area of the same object for a similar amount of time. The last problem to solve is how to modify the task structure in order to explicitly consider bi-manual actions. A possible way is to add a novel kind of subtask, namely a bi-manual subtask, which takes as pre-condition the two post-conditions of the last uni-manual actions. In other words, the bi-manual subtask is executed only if left and right arms have successfully reached the object to manipulate with both hands.

Finally, another aspect to investigate consists in assessing the user-friendliness and intuitiveness of the presented framework for structured task transfer. To this end, an extensive user study can be conducted. The study has to involve either people with a technical or robotic background than inexpert users. To assess the effectiveness of the framework, one can use quantitative metrics like the teaching time and success rate over a certain number of trials, as well as qualitative metrics as a questionnaire [162].

CHAPTER 5

Execution of robotic tasks

This chapter discusses the problem of reproducing a learned skill or task on a real robot. The first section focuses on the reproduction of basic skills and presents an approach for on-line human-aware motion adaptation. The approach exploits dynamical system properties to locally modify the robot's trajectory while preserving the convergence to a given target. Moreover, an algorithm is presented that leverages parallel computation on the graphics processing unit to compute robot–obstacles distances in real-time. The second section of this chapter focuses on structured tasks, showing how the execution of a structured task can be flexibly orchestrated and quickly adapted to changes in the environment and human intervention.

5.1. Reactive planning of robotic skills

This section presents a motion adaptation approach for human–robot coexistence which takes into account: *i)* the human safety, *ii)* the correct task execution, and *iii)* the real-time task replanning according to the human behavior. In order to match these requirements, the two-levels hierarchical architecture in Figure 5.1 is used for real-time and human-aware motion generation. The higher level of the architecture is used to monitor the human. In particular, the *Human Observer* monitors the user status to determine if the human is outside the robot's workspace (*far*) or if the human and the robot share the same working area (*interaction*). During the interaction phase, the user can eventually modify the robot behavior via gestures, forcing the robot to replan the current task. Considering the input from the *Human Observer*, the *Robot Behavior Reshaping* module (the lower level) selects the right behavior from a database of motion primitives, scales down the velocity, and modifies the desired goal position. The lower level leverages stable dynamical systems (DS) for on-line task generation and replanning, and it implements a reactive collision-avoidance approach (*DS Modulation* block) to produce collision-free paths (see Section 5.1.1). Depth data from a RGB-D sensor are used to track the human and measure the distance of the robot from eventual obstacles in the workspace. The *Distance and Normal vector* block executes all the needed computations in real-time, using the approach presented in Section 5.1.2.

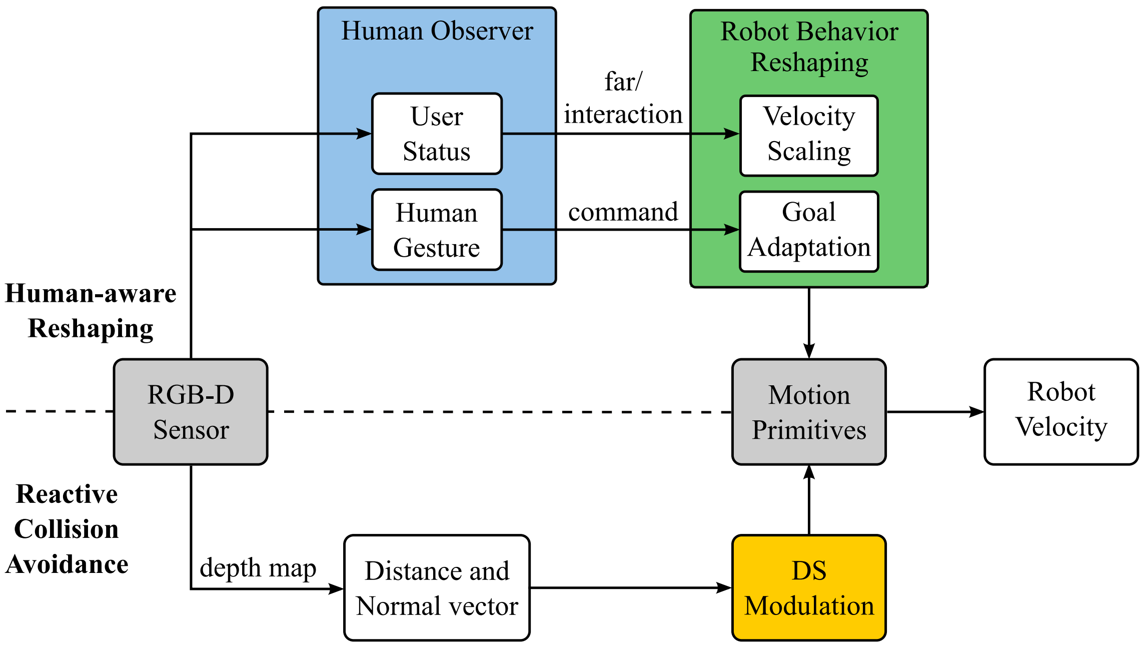


Figure 5.1.: The two-levels architecture used for human-aware motion reshaping. RGB-D data are used to track the human and estimate his status. The same depth data are also exploited to avoid possible collisions with unforeseen obstacles.

5.1.1. Reactive planning using dynamical systems modulation and point clouds

This section describes the approach for real-time motion generation and adaptation using Dynamical Systems (DS). The basic assumption behind the reshaping strategy is that the robot's trajectory is generated using a first-order, globally stable DS

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}), \tag{5.1}$$

where $\boldsymbol{f}(\cdot)$ is a continuous function, $\boldsymbol{x} \in \mathbb{R}^3$ and $\dot{\boldsymbol{x}} \in \mathbb{R}^3$ represent the robot's end-effector position and velocity respectively. Recall that the global asymptotic stability of a DS implies that $\boldsymbol{x} \longrightarrow \boldsymbol{x}^*$ for $t \longrightarrow \infty$, where $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$ (see Appendix A.1). The dynamical system in (5.1) can be designed by an expert user, or learned form demonstrations using one of the approaches presented in Section 3.1. Stable DS are well suited to represent point-to-point motions, since they are guaranteed to convergence to a specified target. Driving robots with DS has also several advantages in terms of robustness to external perturbations, such as unexpected contacts or changes in the goal/initial position. The properties of DS are exploited to realize the real-time reshaping algorithm detailed as follows.

Real-time goal adaptation

Stable dynamical systems can be combined to execute complex robotic tasks, as the prepare coffee task discussed in Section 4.3, consisting of several point-to-point motions. In order to prepare a coffee, the robot has to pick several ingredients and to put them together. In such a task, the DS equilibrium position represents the position of a particular item, or the position where the item has to be placed. Obviously, it is unrealistic to assume fixed positions of the items across several executions of the same task. Hence, the robot has to be able to suddenly adapt to changes in the desired target position. DS are robust to changes in the equilibrium position. Indeed, convergence towards a different goal is achieved by modifying the DS in (5.1) as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x} - \boldsymbol{g}), \tag{5.2}$$

where $\boldsymbol{g}$ is the novel goal (equilibrium) position. It is worth noticing that, if the DS in (5.1) is asymptotically stable, the DS in (5.2) converges to $\boldsymbol{g}$.

Velocity scaling

The severity of possible injuries due to accidental collisions between the human and the robot depends on the velocity of the robot during the impact [60]. Indeed, impacts at high speed have higher probability to generate serious injuries. For this reason, a human-aware planning algorithm has to be able to scale down the velocity of the robot to a safe value [60], without affecting the task execution (i.e., reaching a given target). DS properties can be exploited to realize a human-aware velocity scaling algorithm. In particular, the DS in (5.1) becomes

$$\dot{\boldsymbol{x}} = \alpha_h \boldsymbol{f}(\boldsymbol{x}), \tag{5.3}$$

where $0 < \alpha_{min} \leq \alpha_h \leq 1$ is a scalar function. The value of α_h depends on the human–robot distance. If the user enters the robot's workspace, $\alpha_h = \alpha_{min}$ and the velocity of the robot is reduced to a safe value. When the user is outside the workspace, $\alpha_h = \alpha_{max}$ and the task is executed at maximum speed. α_h is chosen strictly positive in order to preserve the convergence properties of the DS in (5.3). Hence, the robot's velocity is modified to match safety requirements without compromising the task execution.

Dynamical systems modulation for reactive collision avoidance

DS modulation is a reactive approach for collision avoidance that allows to locally modify the trajectory of a DS without affecting its equilibrium points. The aim of the DS modulation is to find a certain state dependent matrix $\boldsymbol{M}(\boldsymbol{x})$ such that the modulated DS

$$\dot{\boldsymbol{x}} = \boldsymbol{M}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}) \tag{5.4}$$

generates collision-free paths. The work in [77] proposes an approach to find a matrix $\boldsymbol{M}(\boldsymbol{x})$ that modifies the robot's velocity in order to avoid possible collisions without affecting the equilibria of the modulated DS. In particular, the modulation matrix $\boldsymbol{M}(\boldsymbol{x})$ in (5.4) generates a collision-free path by reducing the velocity of the robot in the direction normal to the obstacle surface and by projecting the motion into a plane tangent to the obstacle (tangential hyperplane). The approach in [77] exploits an analytical representation of the obstacle surfaces to compute the modulation matrix. The work in [136, 137] extends the approach in [77] by considering the case in which objects are represented via point clouds and an analytical representation of their surfaces is not available. The approaches in [136, 137] are detailed as follows.

This section firstly considers the case where the trajectory of the robot's end-effector has to be modified to avoid collisions with a single obstacle. The case of multiple obstacles, as well as the possibility to avoid collisions with the robot's body, are presented later in this section. Consider that one d-dimensional obstacle is present in the workspace of the robot, for instance, the obstacle close to the end-effector of the robot in Figure 5.2. Assume also that the normal vector to the obstacle surface $\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}}) = [\hat{n}_1(\bar{\boldsymbol{x}}), \cdots, \hat{n}_d(\bar{\boldsymbol{x}})]^{\mathrm{T}}$ is defined for all $\bar{\boldsymbol{x}}$. For each point on the obstacle

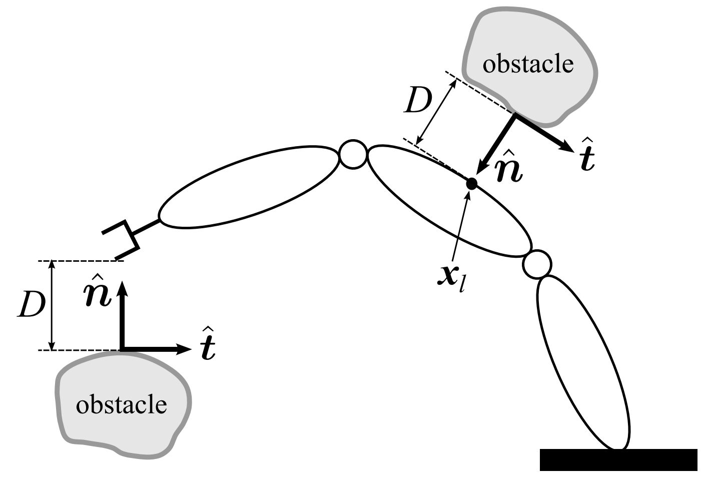


Figure 5.2.: Representation of a robotic manipulator with two obstacles in its workspace. The computation of the modulation matrix requires the distance D between a point on the robot and a point on the obstacle, and the unit vector $\hat{\boldsymbol{n}}$ normal to the surface of the obstacle. In the depicted 2D case, the tangential hyperplane is the unit vector $\hat{\boldsymbol{t}}$ orthogonal to $\hat{\boldsymbol{n}}$ and therefore tangent to the obstacle surface.

surface, the tangential hyperplane is computed as

$$\hat{t}_i^j(\bar{\boldsymbol{x}}) = \begin{cases} -\hat{n}_{i+1}(\bar{\boldsymbol{x}}) & j=1 \\ \hat{n}_1(\bar{\boldsymbol{x}}) & j=i+1 \quad i=1,\dots,d-1, j=1,\dots,d, \\ 0 & j\neq 1, j\neq i+1 \end{cases} \tag{5.5}$$

where $\hat{t}_i^j$ corresponds to the j-th component of the i-th basis vector. Given the definition of the tangential hyperplane in (5.5), it is easy to show that the matrix $\boldsymbol{V}(\bar{\boldsymbol{x}}) = \left[\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}}), \hat{\boldsymbol{t}}_1(\bar{\boldsymbol{x}}), \cdots, \hat{\boldsymbol{t}}_{d-1}(\bar{\boldsymbol{x}})\right]$ represents an orthonormal basis of the d-dimensional space.

Given the robot–obstacle distance $D(\bar{\boldsymbol{x}}_m)$, where $\bar{\boldsymbol{x}}_m$ is the point of minimum distance on the obstacle surface, it is possible to define the matrix $\boldsymbol{E}(\bar{\boldsymbol{x}}_m) = \text{diag}(\lambda_1(\bar{\boldsymbol{x}}_m),\dots,\lambda_n(\bar{\boldsymbol{x}}_m))$, where

$$\begin{cases} \lambda_1(\bar{\boldsymbol{x}}_m) = 1 - \dfrac{1-\varepsilon}{D(\bar{\boldsymbol{x}}_m)+1}, \\ \lambda_i(\bar{\boldsymbol{x}}_m) = 1 + \dfrac{1}{D(\bar{\boldsymbol{x}}_m)+1} \quad i=2,3,\dots,d, \end{cases} \tag{5.6}$$

and $\varepsilon > 0$ is an arbitrary small constant ($\varepsilon = 10^{-5}$ in this work), used to avoid that $\lambda_1(\bar{\boldsymbol{x}}_m) = 0$. A suitable modulation matrix (see Equation (5.4)) is then computed as

$$\boldsymbol{M}(\bar{\boldsymbol{x}}_m) = \boldsymbol{V}(\bar{\boldsymbol{x}}_m)\boldsymbol{E}(\bar{\boldsymbol{x}}_m)\boldsymbol{V}(\bar{\boldsymbol{x}}_m)^{-1}. \tag{5.7}$$

Notice that the modulation matrix in (5.7) is symmetric and positive definite by construction. This implies that the equilibrium points of the modulated DS remain unchanged, since $\boldsymbol{M}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \leftrightarrow \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, $\forall \boldsymbol{x}$. The eigenvalue $\lambda_1(\bar{\boldsymbol{x}}_m)$ in (5.6) tends to ε for $D(\bar{\boldsymbol{x}}_m) \to 0$. Being $\lambda_1(\bar{\boldsymbol{x}}_m)$ the velocity component along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}})$, the robot does not generate motions towards the obstacle if[1] $D(\bar{\boldsymbol{x}}_m) = 0$, while the robot can freely move in the tangential hyperplane, where $\lambda_i(\bar{\boldsymbol{x}}_m) \geq$

[1]The robot's velocity along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}})$ is zero only if $\varepsilon = 0$. Being $\varepsilon = 10^{-5}$ the velocity component along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}})$ is negligible in practical cases.

1. Being the tangential hyperplane orthogonal to the obstacle surface, the modulation matrix locally modifies the DS trajectories and it generates collision-free paths. The modulation matrix is updated at each time step. Hence, in order to avoid collisions with multiple obstacles in the scene, one has to simply avoid the closest obstacle at the current time [136].

The presented formulation of the DS modulation works for static obstacles. In case of a single moving obstacle, the modulated system becomes [137]

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{M}(\tilde{\boldsymbol{x}})(\boldsymbol{f}(\boldsymbol{x}) - \dot{\boldsymbol{x}}_T - \dot{\boldsymbol{x}}_A \times \tilde{\boldsymbol{x}}) + \dot{\boldsymbol{x}}_T + \dot{\boldsymbol{x}}_A \times \tilde{\boldsymbol{x}} = \boldsymbol{M}(\tilde{\boldsymbol{x}})(\boldsymbol{f}(\boldsymbol{x}) - \dot{\boldsymbol{x}}_O) + \dot{\boldsymbol{x}}_O \\ &= \boldsymbol{M}(\tilde{\boldsymbol{x}})\boldsymbol{f}(\boldsymbol{x}) + (\boldsymbol{I} - \boldsymbol{M}(\tilde{\boldsymbol{x}}))\dot{\boldsymbol{x}}_O,\end{aligned} \tag{5.8}$$

where $\dot{\boldsymbol{x}}_O = \dot{\boldsymbol{x}}_T + \dot{\boldsymbol{x}}_A \times \tilde{\boldsymbol{x}}$, $\dot{\boldsymbol{x}}_T$ and $\dot{\boldsymbol{x}}_A$ are the linear and angular velocities of the obstacle respectively, $\boldsymbol{I}$ is the d-dimensional identity matrix, and $\boldsymbol{M}(\tilde{\boldsymbol{x}})$ is calculated using (5.7). The term $\boldsymbol{M}(\boldsymbol{f} - \dot{\boldsymbol{x}}_O)$ is a modulation in the obstacle coordinate system, that guarantees the collision avoidance in the current instant. The additional term $\dot{\boldsymbol{x}}_O$ puts the system in the robot coordinate system and guarantees the collisions avoidance in the following time instant. In case of multiple moving obstacles, the modulated system in (5.8) is simply computed considering the velocity of the closest (most dangerous) obstacle.

As for the static case, the equilibrium point of the modulated DS in (5.8) remains unchanged. In order to prove this result, the velocity of the obstacle $\dot{\boldsymbol{x}}_O$ is assumed continuous and bounded ($0 \leq \|\dot{\boldsymbol{x}}_O\| \leq \dot{x}_{max}$) for all $t \geq t_0$. Notice that these assumptions are naturally verified if there are no impacts between the robot and the obstacles. Moreover, $\boldsymbol{x}^*$ indicates the globally asymptotically stable equilibrium for the original system (5.1). Since the system is globally asymptotically stable, it holds that $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$ only at the equilibrium point $\boldsymbol{x}^*$. Under these assumptions, an augmented version of the original system in the form

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \dot{\phi} \\ \dot{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} -\alpha(\phi - 1) \\ \boldsymbol{f} \end{bmatrix}, \tag{5.9}$$

has a globally asymptotically stable equilibrium at $\boldsymbol{\xi}^* = [1, \boldsymbol{x}^{*\mathrm{T}}]^{\mathrm{T}}$ if $\alpha > 0$. Using (5.9), and assuming that $\phi(0) = 1$, the modulated DS (5.8) can be rewritten as

$$\dot{\boldsymbol{\xi}} = \underbrace{\left[\begin{array}{c|c} \alpha & \boldsymbol{0}^T \\ \hline (\boldsymbol{M} - \boldsymbol{I})\dot{\boldsymbol{x}}_O & \boldsymbol{M} \end{array}\right]}_{\boldsymbol{\Lambda}(\boldsymbol{x},t)} \begin{bmatrix} -\phi(t) \\ \boldsymbol{f} \end{bmatrix} + \begin{bmatrix} \alpha \\ \boldsymbol{0} \end{bmatrix}. \tag{5.10}$$

Considering that the lower block-triangular matrix $\boldsymbol{\Lambda}(\cdot)$ in (5.10) has full rank ($\alpha > 0$ and $\boldsymbol{M}$ positive definite) for all $\boldsymbol{x}$, $t \geq t_0$, it is possible to conclude that the modulated system (5.10) has the same equilibrium of (5.9). Indeed, the velocity $\dot{\boldsymbol{\xi}}$ vanishes only at $\boldsymbol{\xi}^* = [1, \boldsymbol{x}^{*\mathrm{T}}]^{\mathrm{T}}$. This means that the modulation does not affect the equilibrium of the modulated system.

Despite the described algorithm saves the equilibrium points of the modulated DS, the convergence to the goal can be, in some cases, really slow. Consider a static scenario with the robot lying on the obstacle surface, i.e. the robot position $\boldsymbol{x} = \bar{\boldsymbol{x}}$, and assume that the vector field $\boldsymbol{f}(\boldsymbol{x})$ is parallel to the obstacle normal vector. These conditions are mathematically expressed as

$$\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}})^T \frac{\boldsymbol{f}(\bar{\boldsymbol{x}})}{\|\boldsymbol{f}(\bar{\boldsymbol{x}})\|} = \pm 1 \quad \text{and} \quad D(\tilde{\boldsymbol{x}}) = D(\bar{\boldsymbol{x}}) = 0. \tag{5.11}$$

Recalling that $\boldsymbol{V}(\bar{\boldsymbol{x}}) = [\hat{\boldsymbol{n}}(\bar{\boldsymbol{x}}), \hat{\boldsymbol{v}}_1(\bar{\boldsymbol{x}}), \cdots, \hat{\boldsymbol{v}}_{d-1}(\bar{\boldsymbol{x}})]$, it is straightforward to verify that the velocity components along the tangential directions $\hat{\boldsymbol{v}}_i$ are zero. Since $\hat{\boldsymbol{n}}^{\mathrm{T}}\dot{\boldsymbol{x}} \approx 0$ when the robot is on the obstacle surface, the robot will, in practice, stuck in a spurious equilibrium. It is worth noticing that a fixed obstacle is assumed because this problem hardly affects the modulation in (5.8) due to the contribution given by $\dot{\boldsymbol{x}}_O$. An on-line algorithm to escape spurious attractors has been proposed in [77] and summarized in Algorithm 5. First, the algorithm detects when the robot is in a local minimum, checking the conditions $\dot{\boldsymbol{x}}_i \leq \tau_\varepsilon$ and $D(\tilde{\boldsymbol{x}}_i) = 0$, where τ_ε is a constant depending on the chosen ε. Then, a small perturbation γ is applied in one of the tangential directions $\hat{\boldsymbol{v}}_i$, until the robot exits from the basin of attraction of the spurious equilibrium. The positive scalar γ controls the amplitude of the movements along $\hat{\boldsymbol{v}}_i$. Small values of γ reduce the drift due to the integration error (depending on the δt), guaranteeing an effective collision avoidance. However, a very small value of γ highly reduces the avoiding speed and it is not recommended in real-time applications.

Algorithm 5 Avoid slow convergence of the modulated system

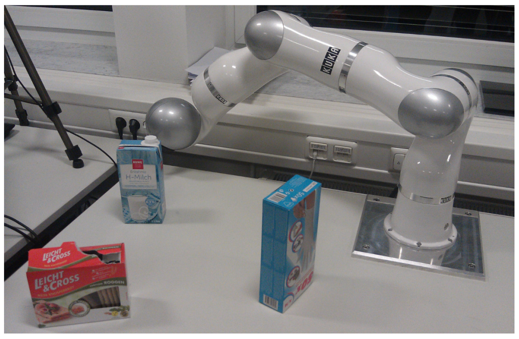
```
Given $\tilde{\boldsymbol{x}}_i$, $\dot{\boldsymbol{x}}_i$ and the integration time step $\delta t$
1.  if $\dot{\boldsymbol{x}}_i \leq \tau_\varepsilon$ and $D_i = 0$ then
2.      Choose one of the tangential directions $\hat{\boldsymbol{v}}_i$
3.      Define a (small) positive scalar $\gamma > 0$
4.      while continue do
5.          $\boldsymbol{x}_{i+1} \rightarrow \boldsymbol{x}_i + \gamma \hat{\boldsymbol{v}}_i \delta t$
6.          Calculate $\dot{\boldsymbol{x}}_{i+1}$ using (5.8)
7.          if $\hat{\boldsymbol{n}}^T \dot{\boldsymbol{x}}_{i+1} > \tau_\varepsilon$ or $\hat{\boldsymbol{v}}_i^T \dot{\boldsymbol{x}}_{i+1} > 0$ then
8.              continue = false
9.          end if
10.         $t \rightarrow t + 1$
11.     end while
12. end if
```

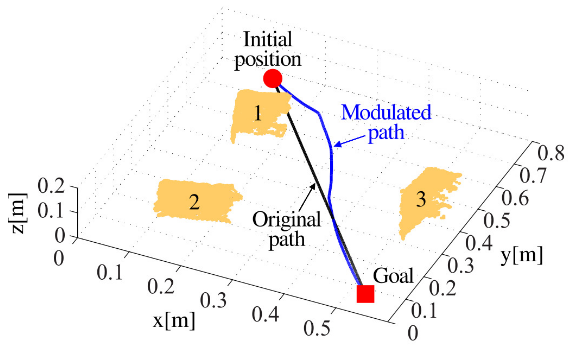
The modulation matrix is previously computed considering the robot as a point-mass object. In order to apply DS modulation in real scenarios, two problems have to be considered. First, the end-effector of the robot is not a point but a rigid body. Second, collisions may occur with other links apart from the end-effector. There are mainly two possibilities to take into account the physical dimensions of the robot links when computing robot–obstacle distances. The first possibility is to cover the robot body with basic primitives like spheres and to compute the distance of the obstacles from the surface of each sphere. The second possibility consists in computing the distance of the obstacles from the surface of each link using a triangular mesh model. This second possibility generates more accurate distance estimations, but it is computationally expensive. An approach for real-time distance computation using mesh models is presented in Section 5.1.2.

Apart from avoiding collisions with the end-effector, real robots have to prevent possible collisions between their links and eventual obstacles in the scene. In order to avoid collisions with the robot body, an approach for link collision avoidance has to be implemented. To this end, this works combines multi-priority inverse kinematics (see Appendix A.2.2 and [142]) with the DS modulation. Roughly speaking, collision avoidance with the robot body is treated as a prioritized task executed in the null-space of the manipulator. Recall that the general solution for the multi-priority inverse kinematics problem is given by

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^\dagger(\boldsymbol{q})\dot{\boldsymbol{x}} + (\boldsymbol{I} - \boldsymbol{J}^\dagger(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\dot{\boldsymbol{q}}_N, \tag{5.12}$$

where $\dot{\boldsymbol{q}}$ is the joint velocity, $\dot{\boldsymbol{x}}$ is end-effector velocity, $\boldsymbol{J}(\boldsymbol{q})$ is the manipulator Jacobian, and $\boldsymbol{J}^\dagger$ denotes the Moore–Penrose pseudoinverse of $\boldsymbol{J}$. The second term in (5.12) projects an arbitrary velocity $\dot{\boldsymbol{q}}_N$ into the robot null-space to generate joint motions that do not affect the end-


(a) Experiment setup



(b) 3D view

Figure 5.3.: The KUKA LWR IV+ avoids collisions with three obstacles. Given the point clouds (yellow points) of the obstacles, a collision-free path to the goal is generated using DS modulation.

effector velocity. In order to avoid collisions with the end-effector, the end-effector velocity $\dot{\boldsymbol{x}}$ in (5.12) is chosen as the reshaped velocity $\dot{\boldsymbol{x}} = \alpha_h \boldsymbol{M}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}-\boldsymbol{g})$. As already mentioned, this guarantees a collision-free path for the end-effector. For the null-space avoidance, it is sufficient to modulate the velocity of the closest point to the robot body $\boldsymbol{x}_l$ (see Figure 5.2). Hence, the avoiding velocity $\boldsymbol{M}(\boldsymbol{x}_l)\dot{\boldsymbol{x}}_l$ is used for the null-space motion, which becomes the joint velocity $\dot{\boldsymbol{q}}_N = \boldsymbol{J}_l^{\dagger}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{x}_l)\dot{\boldsymbol{x}}_l$. The resulting joint velocity command is

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\boldsymbol{J}_l^{\dagger}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{x}_l)\dot{\boldsymbol{x}}_l, \tag{5.13}$$

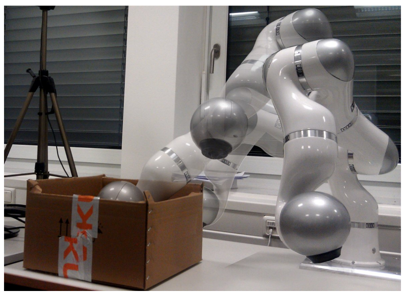
where $\boldsymbol{J}_l(\boldsymbol{q})$ is the Jacobian of $\boldsymbol{x}_l$. The velocity command in (5.13) considers the link collision avoidance as a secondary task which does not affect the end-effector motions. With this approach the end-effector task is correctly executed, since the end-effector convergences to the target and avoids collisions. Moreover, in many situations, the velocity command in (5.13) prevents collisions with the robot body. If collision avoidance is the most important task for a certain application, task priorities can be changed to guarantee that the collision avoidance has always priority [10,48].

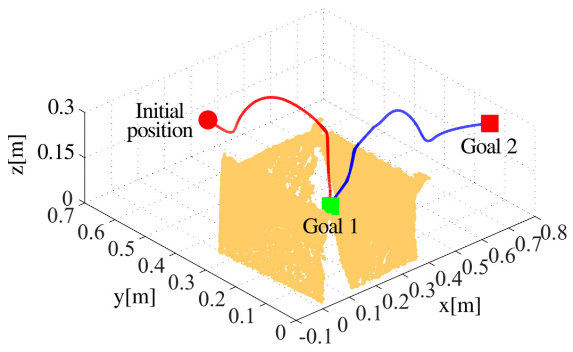
Experimental results

The effectiveness of the presented modulation approach is here tested with simulations and real experiments. A KUKA LWR IV+ robot [15] is used in the real experiments.

Avoiding static obstacles - In this experiment, the robot has to avoid three obstacles in the scene before reaching the target. The situation is shown in Figure 5.3(a). The original path is generated by numerically integrating the DS $\dot{\boldsymbol{x}}(t) = 2(\boldsymbol{g} - \boldsymbol{x}(t))$. The initial position and the goal $\boldsymbol{g}$, together with the dimensions of the obstacles, are chosen so that the robot passes through obstacles and not above them. Figure 5.3(b) shows the obtained results. The distortion of the trajectory due to the modulation is clearly visible already at the beginning of the movement, since the robot starts close to an obstacle. After passing the first obstacle, the modulated trajectory converges to the original one. In the final stage the robot manages to safely pass between objects 2 and 3, placed at 25 cm from each other. Here, the contribution given by 2 and 3 to the modulation matrix tends to balance being the obstacles almost equidistant from the original path.

Going into and out of a box - In this experiment, the robot has to reach two goal positions, one of which is located at the center of a box of size $40 \times 35 \times 20$ cm. One side of the box is


(a) Experiment setup



(b) 3D view

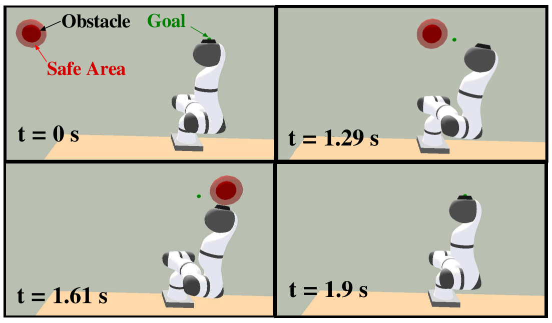
Figure 5.4.: The task of going into and out of a box. Given the point cloud (yellow points) of the box and two goal positions, the robot is driven into (red line) and out (blue line) of a box modulating a switching linear DS.

open. This setup is shown in Figure 5.4(a). The robot starts from a point outside the box, and it is driven towards the first goal $\boldsymbol{g}_1$ by the system $\dot{\boldsymbol{x}}(t) = 2(\boldsymbol{g}_1 - \boldsymbol{x}(t))$. The target $\boldsymbol{g}_1$ is placed inside the box. A collision-free path ending at $\boldsymbol{g}_1$ is generated by modulating the linear DS, as illustrated in Figure 5.3. Having reached $\boldsymbol{g}_1$, the robot is then driven towards the second goal $\boldsymbol{g}_2$ by the system $\dot{\boldsymbol{x}}(t) = 2(\boldsymbol{g}_2 - \boldsymbol{x}(t))$. The target $\boldsymbol{g}_2$ is placed outside the box. As shown in Figure 5.3, DS modulation effectively drives the robot outside the box without any collision, guaranteeing the convergence towards $\boldsymbol{g}_2$. Note that the task cannot be accomplished by using the original DS modulation approach in [77]. In this case, the box should be approximated using a bounding box, thereby preventing the robot to enter. Even if the box is approximated considering each face as a separate obstacle, there may be divisions by zero and the algorithm crashes (see [136] for further details). The presented approach, instead, does not have this problem, because it modulates the system only considering the closest point in each iteration.

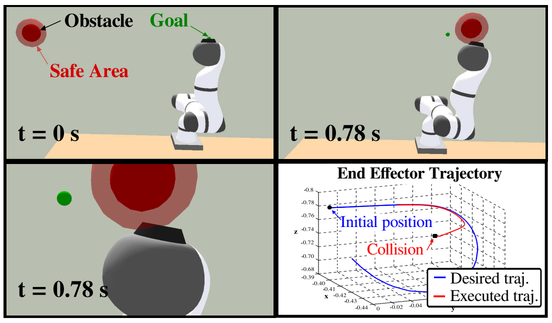
Avoiding dynamic obstacles - In this experiment, the robot has to keep the end-effector in a fixed position while avoiding collisions with a moving spherical obstacle. The initial distance between the robot and the obstacle is 1 m, and the obstacle moves with a constant velocity along a fixed direction (see Figure 5.5). The end-effector trajectory is generated by numerically integrating the linear DS $\dot{\boldsymbol{x}} = 3(\boldsymbol{g} - \boldsymbol{x})$, where $\boldsymbol{g}$ is the goal (initial) position. To consider the physical limitations of the robot, the desired joint trajectories are obtained using inverse kinematics control (see Appendix A.2.2), saturating the joint positions (velocity) that exceed the limits. For the KUKA LWR IV+ robot, used in the simulation, the joint ranges are $\boldsymbol{q}_{max} = -\boldsymbol{q}_{min} = [170, 120, 170, 120, 170, 120, 170]^{\mathrm{T}}$ deg, and the joint maximum velocity $\dot{\boldsymbol{q}}_{max} = -\dot{\boldsymbol{q}}_{min} = [100, 110, 100, 130, 130, 180, 180]^{\mathrm{T}}$ deg/s.

Several tests are performed varying the velocity of the obstacle in the range $[0.5, 1.4]$ m/s. The results in four different time instants, obtained for obstacle velocities of 0.7 and 1.4 m/s, are illustrated in Figure 5.5. The robot is able to avoid the obstacle, coming back in the goal position, until the obstacle velocity reaches 1.4 m/s. For velocities equal or bigger than 1.4 m/s the joint limits are exceeded, and the robot is not able to follow the desired trajectory. The desired trajectory (blue) and the path that the robot follows due to the joint saturations (red) are shown in Figure 5.5(b).

A similar simulation is conducted with the robot reaching a goal while avoiding a fast moving obstacle (see Figure 5.6). Several tests are performed by varying the velocity of the spherical obstacle in the range $[0.5, 1.3]$ m/s. The robot is able to avoid the obstacle until its velocity reaches 1.3 m/s, as shown in Figure 5.6.

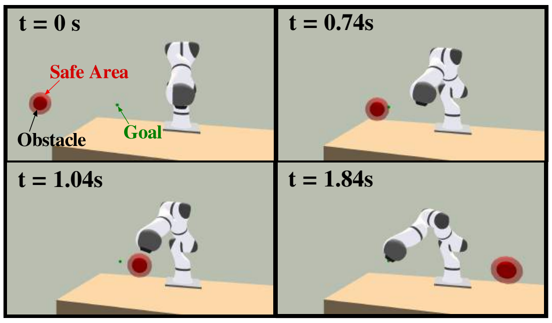


(a) Obstacle velocity $[0, 0.7, 0]^{\mathrm{T}}$ m/s

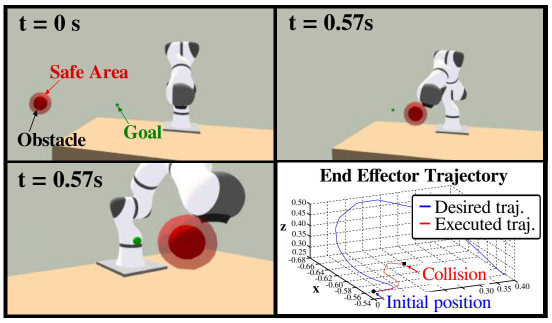


(b) Obstacle velocity $[0, 1.4, 0]^{\mathrm{T}}$ m/s

Figure 5.5.: The robot has to avoid a moving obstacle and return to the initial position. Obstacles slower than 1.4 m/s are properly avoided.



(a) Obstacle velocity $[0, 0.7, 0]^{\mathrm{T}}$ m/s



(b) Obstacle velocity $[0, 1.3, 0]^{\mathrm{T}}$ m/s

Figure 5.6.: The robot has to avoid a moving obstacle and converge to the goal position. The obstacle is avoided until its velocity reaches 1.3 m/s.

In this experiment the robot has to keep the end-effector in a fixed position while the user tries to hit it with the right hand. The desired Cartesian pose is sent to the robot at 1 kHz using the *Fast Reaserch Interface* [139]. The robot trajectory is generated by integrating the linear DS $\dot{\boldsymbol{x}} = 3(\boldsymbol{g} - \boldsymbol{x})$, where $\boldsymbol{g}$ is the goal (initial) position. The human is tracked at 30 Hz using a RBG-D camera and the *OpenNI* library [5]. A constant velocity Kalman filter is used to reduce the noise of the measured hand position and to estimate the hand velocity. The robot is removed from the sensor depth map using a shader-based filter [6]. Snapshots of the experiment, together with robot end-effector and human hand trajectories, are shown in Figure 5.7. The robot is effectively able to avoid the collision and to return to the goal position after approximatively 3 s.

5.1.2. Fast robot-obstacle distance computation using parallel programming

The modulation approach presented in the previous section requires the distance between the robotic manipulator and eventual objects in the scene. As discussed in this section, required distances can be computed by monitoring the scene with a RGB-D camera. Usually, the data given by the visual sensors are firstly transformed into 3D point clouds and then used to compute robot–obstacle distances [12, 112]. However, operating directly on the depth image provided by a depth sensor reduces the computation time, as experimentally shown in [51, 52]. This performance improvement mainly depends on structure of the depth space, which is efficiently exploited to

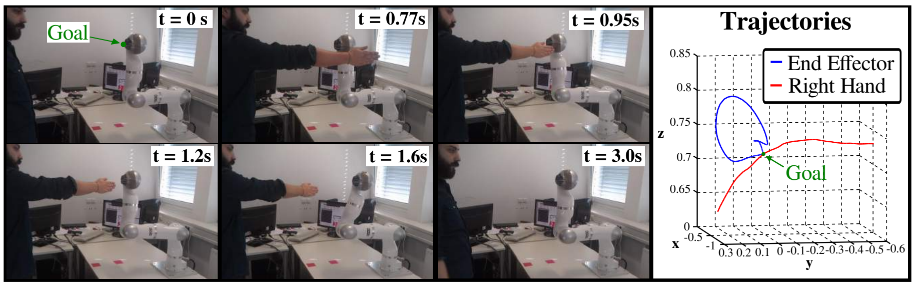


Figure 5.7.: The robot avoids collisions with the human's right hand and returns to the initial position. The norm of the hand velocity is in the range $[0.45, 0.6]$ m/s.

speed-up the computation. This section discusses the problem of robot–obstacle distance calculation in the depth space. First, the structure of the depth space is briefly described, and the approach in [52] that directly computes distances in the depth space is presented. Then, an approach for fast Distance Evaluation in the Depth space (fDED) is described.

Depth space structure

The depth space is a non-homogeneous 2.5-dimensional space. Indeed, depth sensors represent a Cartesian point $\boldsymbol{x} \in \mathbb{R}^3$ with three non-homogeneous values. Two values are the coordinates of the projection of $\boldsymbol{x}$ in the depth sensor plane, indicated with the subscripts x and y in this work. The third value is the distance (depth) of the point $\boldsymbol{x}$ from the sensor plane. A generic Cartesian point expressed in a world frame $\boldsymbol{x}_w = [x_w, y_w, z_w]^\mathrm{T}$ can be expressed in the depth sensor frame by applying the homogeneous transformation $\boldsymbol{x}_c = \boldsymbol{R}\boldsymbol{x}_w + \boldsymbol{t}$. Using the pin-hole camera model [96], $\boldsymbol{x}_c = [x_c, y_c, z_c]^\mathrm{T}$ is mapped into the depth point

$$\boldsymbol{x}_d = \begin{bmatrix} p_x \\ p_y \\ d_p \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}, \tag{5.14}$$

where p_x and p_y are the pixel coordinates of the projected point, f_x and f_y are the focal lengths of the camera in pixel units, c_x and c_y are the pixel coordinates of the camera's optical center. As illustrated in Figure 5.8, projecting Cartesian points in the depth space generates occlusions, the so-called gray area. This is because a point $\boldsymbol{x}_c$ with depth d_p occludes all the points having depth $d \geq d_p$ and lying on the virtual line connecting the camera center and $\boldsymbol{x}_c$. The gray area is thus a region of uncertainty, since no information on this region can be stored in the depth space representation.

Computing the Cartesian distance in the depth space

Given the depth representations of a point on the robot $\boldsymbol{x}_{R_d} = [r_x, r_y, d_r]^\mathrm{T}$ and a point in the scene $\boldsymbol{x}_{O_d} = [o_x, o_y, d_o]^\mathrm{T}$, the distance between $\boldsymbol{x}_{R_d}$ and $\boldsymbol{x}_{O_d}$ is [52]

$$d(\boldsymbol{x}_{R_d}, \boldsymbol{x}_{O_d}) = \sqrt{(v_x)^2 + (v_y)^2 + (v_z)^2}, \tag{5.15}$$

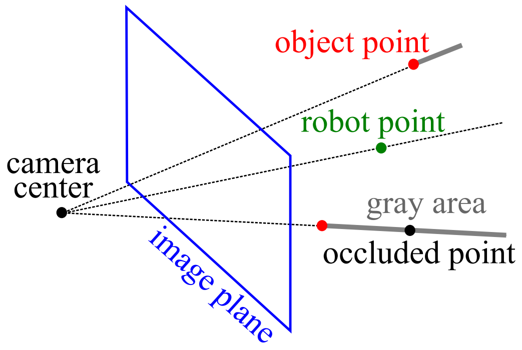


Figure 5.8.: Representation of the gray area generated by projecting a 3D point into the image plane.

where

$$\begin{aligned} v_x &= \frac{(o_x - c_x)d_o - (r_x - c_x)d_r}{f_x}, \\ v_y &= \frac{(o_y - c_y)d_o - (r_y - c_y)d_r}{f_y}, \\ v_z &= d_o - d_r. \end{aligned} \tag{5.16}$$

To take into account the gray area, the depth value d_o of the object point $\boldsymbol{x}_{O_d}$ is changed to

$$d_o = \begin{cases} d_r & \text{if } d_r \geq d_o, \\ d_o & \text{otherwise} \end{cases}. \tag{5.17}$$

In other words, if the robot point is behind the scene point ($d_r > d_o$), the depth of the object point is $d_o = d_r$ to consider possible occluded points (see Figure 5.8).

Fast distance evaluation in the depth space

The distance of each object point from each link of the robot can be computed by iteratively applying Equation (5.15). To reduce the computation time, the robot body can be approximated using simple geometric primitives, like cylinders or spheres. This approach is used, for instance, in [51, 52]. In order to provide more accurate and realistic distance estimations, a triangular mesh (CAD) model of the robot is exploited in this work. Moreover, distances are computed in real-time exploiting the fDED (fast distance evaluation in the depth space) approach proposed in [134]. The main idea of fDED is to reduce the computation time by reducing the number of points considered for distance estimation. To this end, a lattice of robot and a lattice of objects points are used. The lattices are directly computed in the depth space. The resulting algorithm is highly parallelizable, which allows the parallel execution of the needed computations on the Graphics Processing Unit (GPU). An overview of the fDED approach is shown in Figure 5.9.

The raw depth image from the sensor (real depth) contains also points belonging to the robot surface. It is clear that points on the manipulator should not be treated as obstacles to be avoided. The robot can be efficiently removed from the depth image given its mesh model and its current joint configuration (see the top panel in Figure 5.9). First, the mesh model of the robot is projected into the depth space to create a depth map which contains only robot points, the so-called *virtual depth*. The projection is computed by applying the transformation (5.14) to each vertex of the robot model. Being the robot model composed by one mesh model for each link, an unique label (index)

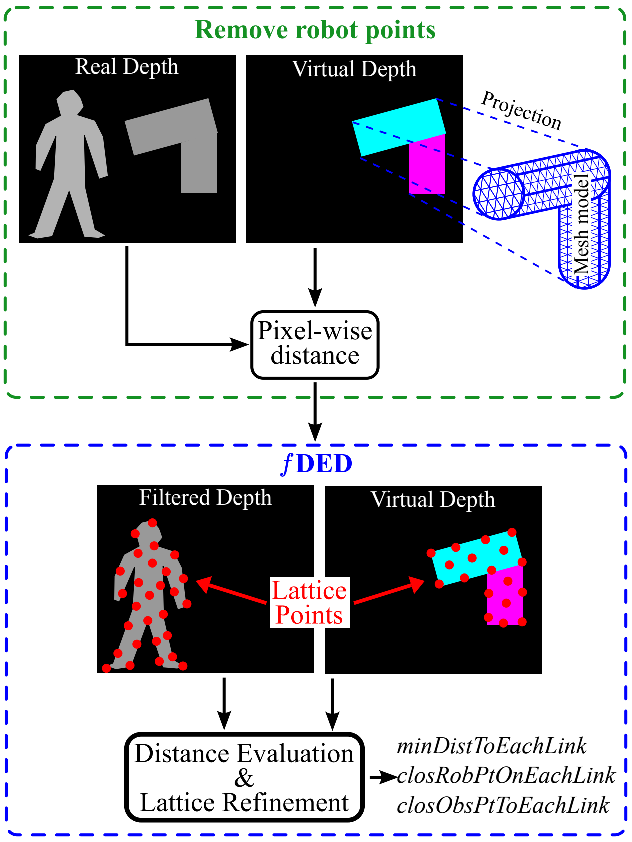


Figure 5.9.: Overview of the fast distance evaluation in the depth space (fDED) approach.

is attached to each point. The index is used to determine to which link a certain point belongs to. After this step, each point on the robot $\boldsymbol{x}_R$ is converted into a depth point $\boldsymbol{x}_{R_d} = [r_x, r_y, d_r]^{\mathrm{T}}$. As previously discussed, the position of $\boldsymbol{x}_{R_d}$ in the depth plane is determined by the pixel coordinates r_x and r_y. Given the virtual depth, the robot is removed from the real depth map through a pixel-wise comparison. More specifically, the depth value d of the point located at (r_x, r_x) is extracted from the real depth map and compared with the relative robot depth value d_r. If $|d - d_r| \leq \delta$, the point belongs to the robot and it is removed from the real depth map. The value $\delta = 5\,\mathrm{cm}$ is used in this work. The pixel-wise comparison is repeated for all the points in the virtual depth map, obtaining the filtered depth map in Figure 5.9 with the robot removed. Pixel-wise comparisons are executed separately for each point. A GPU implementation [6] of the described approach takes less than 1 ms to remove the robot from the depth image.

Computing the pair-wise distance of every point in the virtual depth from every point in the filtered depth is computationally unfeasible in real-time ($\approx 1\,\mathrm{ms}$). For this reason, fDED leverages an approach that drastically reduces the number of considered points without significantly affecting the accurate distance evaluation. Instead of considering all the robot points, they are sub-sampled by building a lattice of robot points. The lattice of robot points has to satisfy two requirements: *1)* the lattice has to cover the entire robot, and *2)* the lattice has to be fine enough to produce accurate results. To fulfill both requirements and reduce the robot points, the following procedure is adopted:

I. A (quasi-) equally spaced, coarse lattice of robot points is created (see Figure 5.10). The lattice covers the entire robot.

II. The distances of the robot lattice points to all objects in the scene are calculated.

III. For each link, the closest robot lattice point $\boldsymbol{x}^*_{R_l}$ to an object is determined.

The lattice of robot points is created by dividing the virtual depth into squared tiles, as shown in Figure 5.10. The initial side length T_s of the tiles is a tunable parameter. In each tile, a different thread checks if there are robot points. The robot point closest to the center of the tile is selected as a lattice point. Each tile can contain at most one lattice point for each link. Tiles that do not contain any robot point are simply dropped out. After the closest robot lattice point $\boldsymbol{x}^*_{R_l}$ to an object is determined (step III), steps II and III are repeated for all the points in the tile around $\boldsymbol{x}^*_{R_l}$. This refinement step is needed to increase the accuracy, otherwise bounded by the size of the tile. This procedure can be executed in parallel for each lattice point, which allows a GPU implementation.

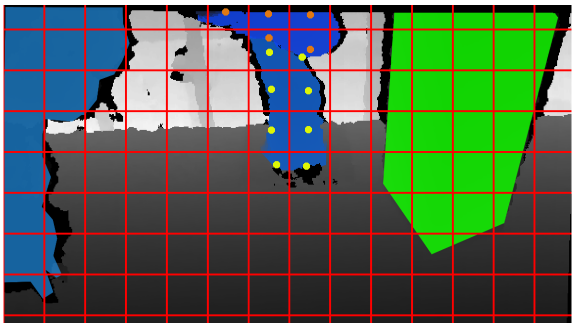
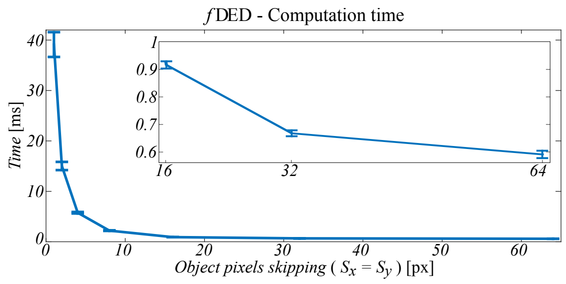


Figure 5.10.: The creation of the lattice of robot points. The blue are points on the robot surface, while the green are points on the object surface (a box). Both the robot and the box are standing on a table, which is not considered as an obstacle. Yellow bullets are the lattice points of the first link, orange bullets the lattice points of the second link.

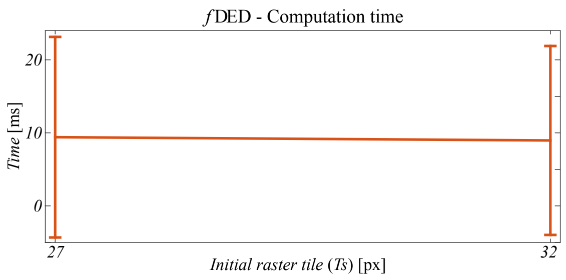
The algorithm used to create the lattice of robot points assumes that the pixel coordinates of each robot point are known a priory. This assumption is not valid for the filtered depth map, since it is not know a priory which pixels represent the obstacles. Moreover, in order to apply the refinement procedure to object points, one should create a lattice of object points for each robot link. Instead of inspecting all the points in the filtered depth map, object points are skipped to increase the execution time. The lattice of object points is simply created by skipping pixels in the filtered depth. The spacing between the object lattice points, namely the step size S_x and S_y, is a tunable parameter and it can vary to produce more or less accurate results. Given an object lattice point with pixel (px) coordinates (o_x, o_y), the next lattice point lies outside the region $[o_x + S_x, o_y + S_y]$. $S_x = S_y = 1\,\text{px}$ means that all the points in the filtered depth map are lattice points. As for the lattice of robot points, the algorithm to create a lattice of objects points can be parallelized and executed on a GPU.

The dynamical system modulation approach presented in Section 5.1.1 requires also the normal vectors at the points of minimum distance. In this work, the normal vector is estimated using a parallel implementation of the algorithm in [100]. Given the point of minimum distance $\boldsymbol{O}_m$ with pixel coordinates (o_x, o_y), a plane π_o tangent to $\boldsymbol{O}_m$ is created using weighted least squares. L points[2] in the neighborhood of (o_x, o_y) are transformed into Cartesian points and used to generate π_o. The normal vector to the plane π_o represents the normal vector at the point $\hat{\boldsymbol{n}}(\boldsymbol{O}_m)$. This algorithm is robust to noise and it requires neither a smooth surface nor a certain density of points.

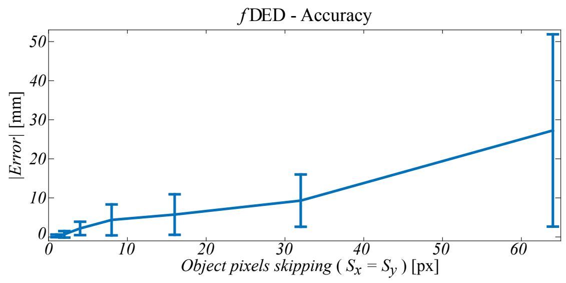
[2]In the experiments, L is set to $L = 1\%$ of the number of points as suggested in [136].



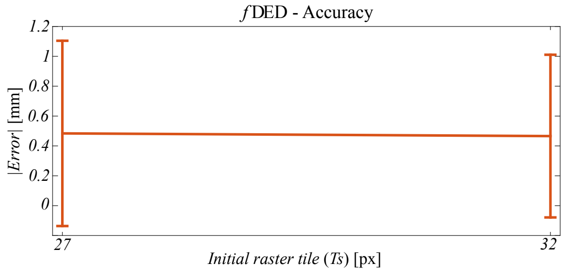
(a) All robot points considered.



(b) All filtered depth points considered.



(c) All robot points considered.



(d) All filtered depth points considered.

Figure 5.11.: Performance of *f*DED in terms of execution time and accuracy (mean and standard deviation over 100 iterations). (a) Dependency of the execution time on the step size. (b) Dependency of the execution time on the initial raster tile. (c) Dependency of the accuracy on the step size. (d) Dependency of the accuracy on the initial raster tile.

Performance evaluation

An experimental evaluation is conducted to show the real-time capabilities (up to 1 ms) of *f*DED and to provide a guideline to tune the parameters used to create the point lattices. The evaluation is conducted using a static scene where robot and obstacles are fixed. The scene is shown in Figure 5.10. A static scene is preferred in order to test the algorithm always in the same conditions. A dynamic environment is considered in the case study in Section 5.1.3. *f*DED is implemented in C++ and runs on a Personal Computer with an Intel® Core^{TM} $i7-4790K$ - 4 Cores CPU, a GeForce GTX 660 - 960 Cores GPU, and 16 GB of memory.

A Guideline for tuning parameters selection - This experiment aims at showing the effects of different parameter sets on the computation time and the accuracy of *f*DED. Understanding the role of each parameter, in fact, is essential to provide a guideline for the parameters selection. Recall that *f*DED has two tunable parameters, i.e. the initial tile size T_s, and the step size (S_x and S_y). The experiment aims at verifying the following hypothesis:

H_1. Increasing (decreasing) T_s reduces (increases) the computation time and the accuracy.

H_2. Increasing (decreasing) S_x and S_y reduces (increases) the computation time and the accuracy.

In order to consider the effects of noisy data while keeping the same (light) conditions, 100 depth images are recorded, and averaged results are given. The robot is a 7 degrees-of-freedom KUKA LWR IV+ [15]. The triangular mesh of the robot has 4861 vertexes, while the depth images have 640×480 pixels (px).

Several analysis are conducted with different values of the tuning parameters. Main obtained results are illustrated in Figure 5.11. For comparison, consider that a brute force approach which considers all the points takes about 128 s on the CPU and about 0.9 s on the GPU to compute the distance. The brute force algorithm, in fact, has to compute 1.5×10^9 distances in order to find the minimum one. Being the most accurate approach (all points are considered), results of the brute force algorithm serves as ground truth for the accuracy. To give more compact results, average errors over the 7 links are used in Figure 5.11.

As shown in Figure 5.11(a) and 5.11(c), reducing S_x and S_y increases the computation time (note that times are in *ms*) and the accuracy, since more points have to be considered. Values of S_x and S_y bigger than 32 px penalize too much the accuracy without significantly reducing the computation time. The initial raster tile value T_s slightly affects the computation time and the accuracy, as shown in Figure 5.11(b) and 5.11(d). Obtained results allow to conclude that hypothesis H_1 and H_2 are verified. Taking into account the conducted analyses, a parameter set that guarantees an execution time of about 1 ms and an accuracy of about 5 mm is shown in Table 5.1. It is worth noticing that 5 mm of error is widely tolerable considering the typical accuracy of depth sensors [80].

Table 5.1.: A parameter set for *f*DED to obtain $T \approx 1\,\text{ms}$ and $A \approx 5\,\text{mm}$ ("T" computation time, "A" accuracy).

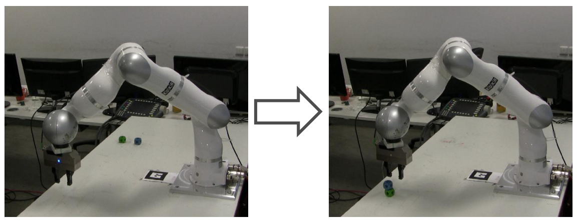
T_s [px]	S_x/S_y [px]	T [ms]	A [mm]
32	16/16	0.9	5

Computation time in the worst case - The worst case scenario for the computation time is obtained by creating a real depth image in which all the 640×480 pixels consist of object points. The robot is located behind those points, in order not to filter any values out of the real depth image. Moreover, the robot covers the entire virtual depth image. Hence, both depth images have to be entirely filled with a lattice of points and the number of distance computations is maximal. In this case, the brute force algorithm computes $(640 \times 480)^2 \approx 0.9 \times 10^{12}$ distances. Time results for different parameter sets are shown in Table 5.2. An average time (over 100 iterations) of 2.8 ms is obtained with $S_x = S_y = 32\,\text{px}$.

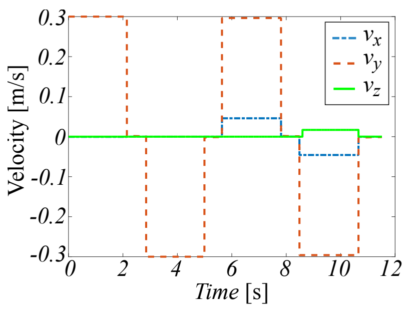
Table 5.2.: *f*DED - Computation time in the worst case scenario.

T_s [px]	S_x/S_y [px]	Robot Pts (%)	Time (mean/std) [ms]
32	8/8	100	6.3/0.19
32	16/16	100	3.7/0.16
32	32/32	100	**2.8/0.16**
32	32/32	50	**1.9/0.16**
32	32/32	30	**1.3/0.16**

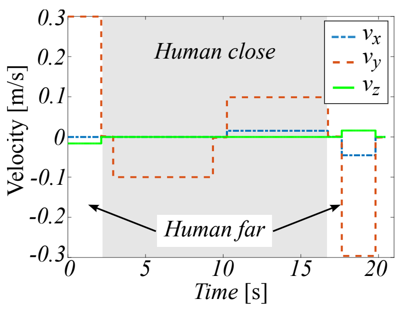
The proposed worst case represents a theoretical upper bound for the computation time, but it is unrealistic in real scenarios. Indeed, having a depth map completely covered by objects is possible in extremely cluttered environments, while assuming that the robot covers all the virtual depth is unrealistic. Camera sensors, in fact, are used to monitor the robot's workspace. If the robot covers all the scene, this means the sensor is not correctly placed [50]. More realistic upper bounds on the computation time are given in Table 5.2 by considering that the robot occupies 50% and 30% of the total points. As for the worst case, the robot is located behind the objects, in order not to filter any values out of the depth image.



(a) Task execution.



(b) Nominal velocity ($\alpha_h = 0.3\,\mathrm{m/s}$).



(c) Velocity scaling.

Figure 5.12.: Execution of the stacking task. (a) Snapshots of the task execution. (b) Robot velocity when the human is not in the workspace. (c) Velocity scaling when the human enters/leaves the workspace.

5.1.3. Evaluation in a human–robot interaction scenario

The architecture for on-line motion replanning in Figure 5.1 is tested in a human–robot interaction scenario. The human is tracked at 30 Hz using a Microsoft Kinect RGB-D camera. The same camera is used to monitor the scene and compute the distance from eventual obstacles. The robot is a 7 degrees-of-freedom (DoF) KUKA LWRIV+ [15], controlled at 500 Hz. As shown in Figure 5.12(a), in this experiment the robot has to stack two blocks. The blocks are initially placed at $\boldsymbol{g}_1 = [-0.55, 0.35, 0.01]^{\mathrm{T}}$ and $\boldsymbol{g}_2 = [-0.45, 0.35, 0.01]^{\mathrm{T}}$ respectively. The stacking positions are $\boldsymbol{g}_3 = [-0.55, -0.3, 0.01]^{\mathrm{T}}$ and $\boldsymbol{g}_4 = [-0.55, -0.3, 0.035]^{\mathrm{T}}$ respectively. The stacking task consists of four point-to-point motions generated by combining four stable DS in the form

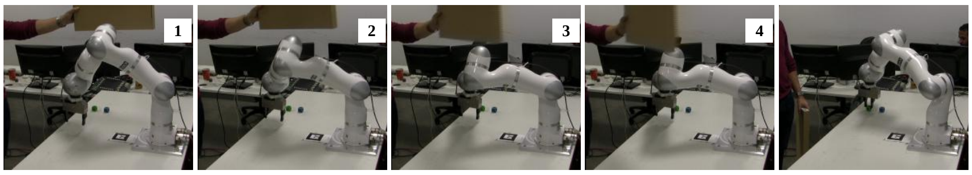
$$\dot{\boldsymbol{x}} = \alpha_h \frac{\boldsymbol{g}_i - \boldsymbol{x}}{\|\boldsymbol{g}_i - \boldsymbol{x}\|},\ i = 1,..,4,$$

where $\alpha_h > 0$ is the desired (constant) speed. The orientation of the robot is kept fixed during the motion. This means that the robot can exploit 1 DoF for null-space collision avoidance.

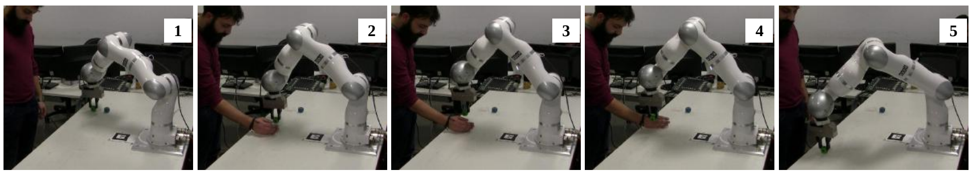
Velocity scaling and collision avoidance

The user can enter the robot workspace at any time and start the interaction. As shown in Figure 5.12(c), when the human is close to the robot, the robot's speed is scaled to $\alpha_h = 0.1\,\mathrm{m/s}$. When the human is far from the robot, the robot moves at the nominal speed of $\alpha_h = 0.3\,\mathrm{m/s}$ (see Figure 5.12(b)). In both cases the robot is able to execute the stacking task.

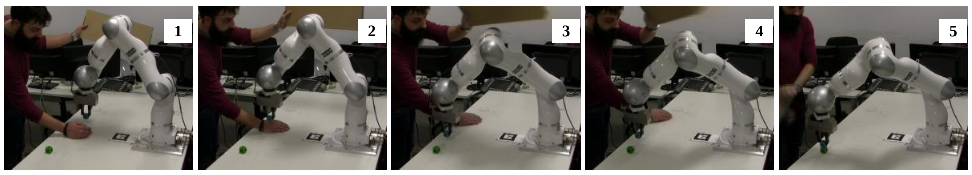
In realistic scenarios the robot needs to avoid possible collisions with dynamic obstacles, preferably without compromising the task execution. Figures 5.13(a)-(c) show the robot avoiding collisions with several moving objects, while correctly executing its task. The robot is able to prevent multiple collisions with its entire body, using the combination of multi-priority inverse kinematics and DS modulation presented in Section 5.1.1.



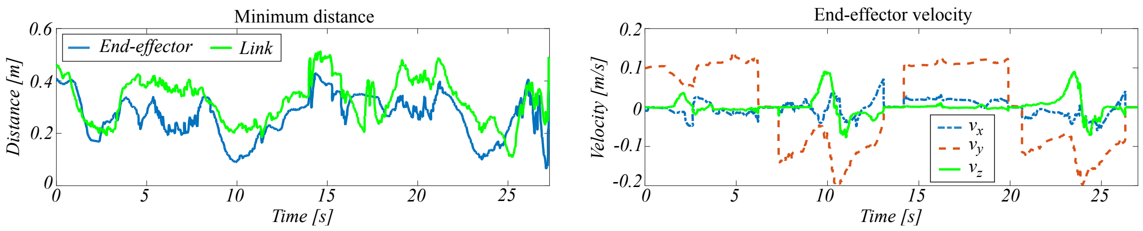
(a) Avoiding collisions between the robot body and a moving obstacle.



(b) Avoiding collisions between the robot end-effector and a moving obstacle (user's hand).



(c) Avoiding multiple collisions with moving obstacles.



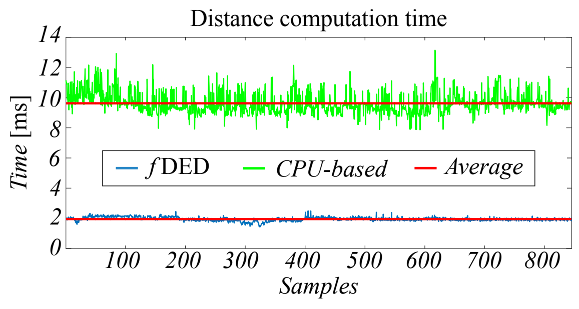
(d) Typical minimum distance and velocity profiles during the interaction with the user.

Figure 5.13.: The robot avoids several collisions with dynamic obstacles while correctly executing a stacking task.

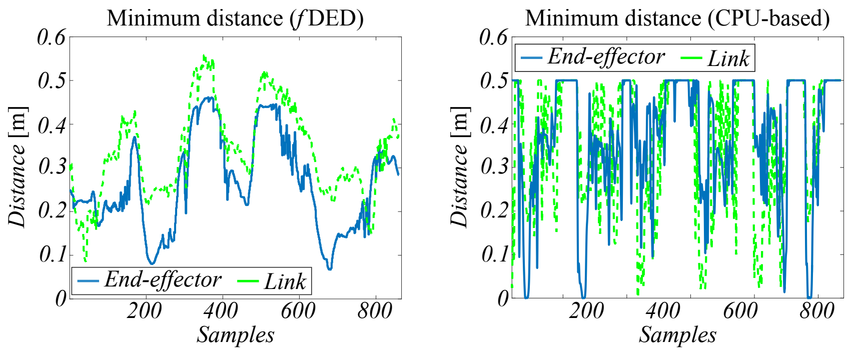
The left panel in Figure 5.13(d) shows the minimum distance between the robot and the closest obstacle. The minimum distance is always bigger than 6 cm, meaning that the robot is effectively able to avoid collisions. The right panel in Figure 5.13(d) shows the end-effector velocity during the interaction. As detailed in Section 5.1.1, the velocity is reduced in the direction towards the obstacle, and increased in the collision-free directions. This generates the avoiding motion while minimizing the impact force (almost zero velocity) in case of unexpected collisions.

Comparison with state-of-the-art approach

This experiment compares the performance of the fDED approach presented in Section 5.1.2) and the CPU-based approach for distance evaluation presented in [52] and used in [138]. The approaches are compared considering the computation time and the minimum distance between obstacles and robot during the task execution. The approach in [52] approximates the robot body with 10 spheres of radius $r_s = 0.12\,\text{m}$, and computes the distance between the points in the depth map and each sphere. The approach is implemented in C++. To further reduce the computation time, the approach in [52] considers a region of surveillance with $\rho = 0.4\,\text{m}$, i.e., it skips all the depth points outside a cube of side 2ρ around each sphere. Points whose depth value is too close ($< 1.8\,\text{m}$) and to far ($> 3.5\,\text{m}$) are also skipped in this work. If no points are found in the



(a)



(b) (c)

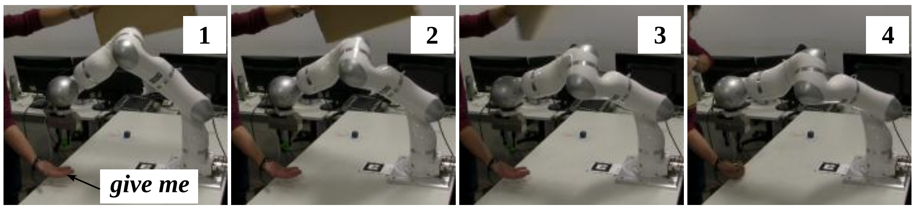
Figure 5.14.: Comparison between *f*DED and the CPU-based approach in [52]. (a) *f*DED spends on average 1.9 ms to compute the distances, while the CPU-based approach requires 9.9 ms. (b)-(c) The robot is more reactive when *f*DED is used to compute the distance and it effectively avoids possible collisions.

neighborhood of the robot, the distance is set to a default value (0.5 m) which does not generate avoiding movements. As for the *f*DED approach, the robot is removed from the sensor depth using the filtering approach previously described.

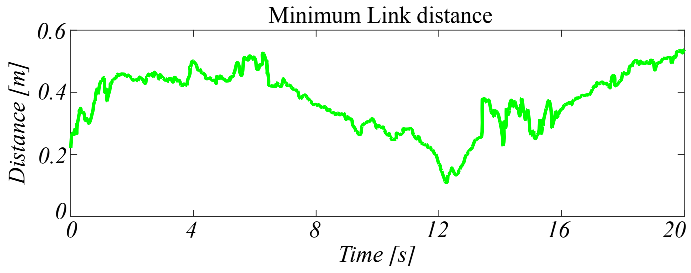
Results in Figure 5.14(a) show that *f*DED takes, on average, 1.9 ms to compute the distances, while the CPU-based approach requires 9.9 ms. The significantly smaller computation time of *f*DED is beneficial when avoiding collisions with moving obstacles. The robot, in fact, reacts quickly to the approaching obstacle and effectively avoids the collision. This result is illustrated in Figure 5.14(b), which shows that the minimum distance between robot and obstacles is always bigger than 6.7 cm. On the contrary, the delay introduced by the CPU-based approach makes the robot less reactive while avoiding possible collisions. As shown in Figure 5.14(c), in some cases the robot fails to avoid the collision (distance equal to zero). Presented results allow to conclude that, in dynamic environments, a fast approach like *f*DED is beneficial to effectively avoid collisions. The drawback of *f*DED is that its implementation requires a dedicated hardware (GPU) for parallel processing.

Cooperative task execution via human gestures

This experiment exploits the architecture described in Section 5.1 to execute the stacking task in cooperation with the human. To this end, a gesture recognition module is added to the *Human Observer* (see Figure 5.1) to modify the task execution via gesture commands. Human gestures are recognized using the motion descriptor proposed in [135] and described in Section 3.2.1. This descriptor has the advantage of being invariant to roto-translations and scaling factors, which makes



(a) Snapshots of the handover task execution.



(b) Minimum robot–obstacle distance during the *handover* task.

Figure 5.15.: The robot gives the picked item to the user and avoids collisions with a dynamic obstacle.

it well-suited to recognize motions performed by different subjects or observed from different views. The approach works by transforming the trajectory of the human torso, right shoulder, elbow and hand into a sequence of invariant values. As already mentioned, the human is tracked at 30 Hz. Continuous gestures are recognized using a sliding widow of length 50 frames, and dynamic time warping [131] for classification. Three gestures are used in this experiment: *i) come* to start the task execution, *ii) stop* to suspend the execution, and *iii) handover right* to command the robot to give the item to the user and continue the execution (take the next item). The 3 gestures are a subset of the 6 gestures used in [135] for continuous gesture recognition. The recognition rate on the full set of 6 gestures is 91.67%.

In order to execute human commands, dynamical systems are exploited to replan the motion in real-time. In particular, if the robot receives a *stop* command, the desired speed is set to $\alpha_h = 0\,$m/s, and the robot waits for other commands. If the robot receives a *come* command, the speed is set to 0.1 m/s (close user) or 0.3 m/s (far user), and the robot can continue the task execution. If the robot receives a *handover right* command, the goal is changed to reach the current hand position (plus an offset of 5 cm along the vertical direction) and leave the item. During the *handover* action, the end-effector avoidance is disabled since the robot has to reach the hand. The null-space avoidance, instead, is active to make the robot able to avoid collisions with its body. As shown in Figure 5.15, the robot successfully executes the *handover* task and, at the same time, avoids the collision with a moving obstacle.

5.2. Execution of structured tasks

Section 4.3 presented an integrated framework for intuitive kinesthetic teaching. The framework combines the benefits of intuitive skill transfer mechanisms, like kinesthetic teaching or imitation learning, and an attentional supervisory system to learn structured tasks from human demonstration. The same framework, with minor changes, is used to learn both uni- and dual-arm structured tasks. This section focuses on exploiting the framework presented in Section 4.3 to execute the learned structured task on real robots. In particular, this section describes how the main blocks of the framework, namely the robot manager (see Section 4.3.1) and the attentional system (see Section 4.3.2), are used to reproduce the task. Moreover, the section presents several experimental results which demonstrates the effectiveness of the discussed approach.

5.2.1. Robot manager for task execution

The Robot Manager (RM) handles low-level aspects of the human–robot interaction and it is responsible for a correct task execution. As discussed in Section 4.3.1, during the teaching phase the RM allows kinesthetic teaching (or imitation learning), automatic segmentation of the demonstrated task, and motion primitives learning. During the execution, the RM has to retrieve the desired robot's trajectory from the learned motion primitives. In particular, the RM receives the next action to execute from the supervisory attentional system (see Section 4.3.2). The commanded action is a symbol, i.e. a unique label, that the RM matches with the associated motion primitive. The RM generates a set of Cartesian poses from the learned motion primitive, which are directly sent to the low-level robot controller. It is worth noticing that the user can decide, at any time, to start a novel teaching session using a dedicated speech command. In this case, the RM switches from an autonomous execution mode to the teaching mode, i.e. the RM activates a zero gravity control (see Appendix A.2.3) for kinesthetic teaching, or a motion re-targetting kinematic control (see Section 4.2) for imitation learning.

5.2.2. Attentional system for task execution

The result of the learning process is a tree that represents the structured task (see, for example, Section 4.3.3). The attentional system stores the entire structure in its long term memory, which is a knowledge base containing all the learned tasks (see Section 4.3.2). During the execution, the *alive* process instantiates the task (or part of it) in the Working Memory (WM), using the retrieve/expansion process shown in Figure 5.16 and detailed in Section 4.3.2. The procedure is analogous for teaching and execution modes up to the subtasks level. During the teaching, in fact, atomic actions are unknown and are attached to the most emphasized subtask. On the contrary, during the execution, atomic actions are effectively commanded to the robot in order to execute the task. Hence, *alive* further decomposes the most emphasized subtasks and commands to the robot manager the most emphasized action with a *true* releaser. Note that the expand/retrieve process is continuously executed. In this way, the planned task is quickly adapted if changes in the environment move the emphasis towards another subtask. This flexibility is exploited, for example, to perform the task execution in cooperation with a human operator, as discussed in the following section.

5.2.3. Execution of uni-manual structured tasks

Experiments in this section show that the presented framework can be effectively applied to *i)* quickly learn and autonomously execute structured tasks, *ii)* execute learned tasks in cooperation with the human, and *iii)* reuse the acquired knowledge in different contexts. To this end, two typical tasks of a kitchen scenario are considered, namely prepare coffee and prepare tea. The robot is a KUKA LWR IV+ [15], equipped with a WSG50 2-fingers gripper. As shown in Figure 5.17(a), objects are recognized and tracked using markers and a RGB-D camera as in [56]. The marker close to the robot base is employed to retrieve the coordinate transformation between the camera frame and the robot base. Due to possible marker occlusions during the teaching, the robot–camera transformation and the pose of the cup are computed at the beginning of each experiment and kept constant during the execution. All the other objects, instead, are continuously

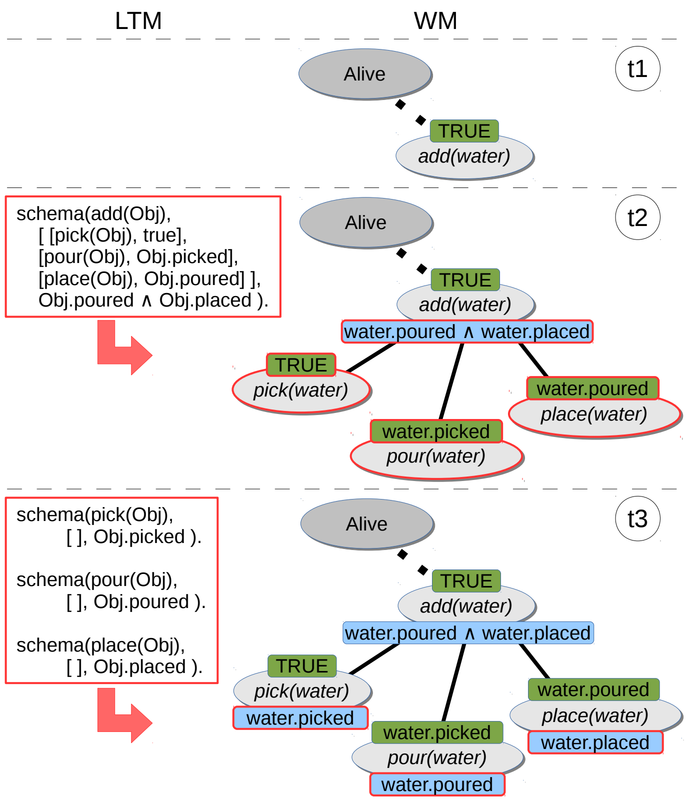
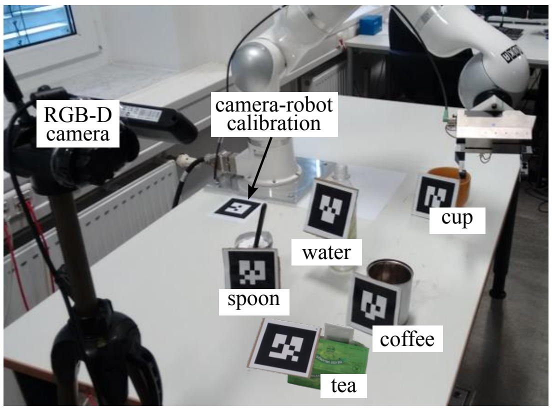


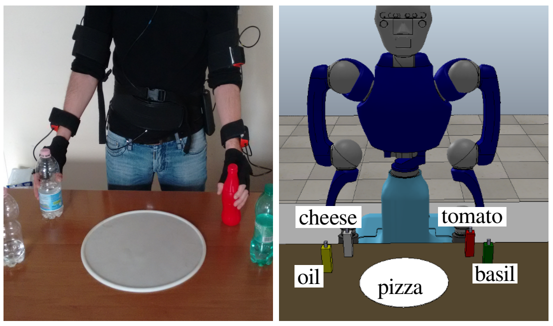
Figure 5.16.: *Alive* manages the retrieve/expansion process used to instantiate part of a task in the working memory. (t1) *Alive* allocates the new node *add(water)*. (t2)-(t3) The schema associated to *add(water)* is selected from the LTM (retrieve phase) and used to hierarchically decompose the node in the WM. In particular, the abstract or concrete sub-nodes contained in the schema are added to the WM and instantiated (expand phase).

tracked at 30 Hz. The user initiates a kinesthetic teaching session via the speech command *teach*. A zero gravity control is used to easily guide the robot in teach mode. The teaching session is terminated by the speech command *done*. The user can interrupt/restart the execution of a learned task using the speech commands *stop/repeat*. During the execution, the RM sends the reference trajectory to the robot low-level controller. The KUKA LWR IV+ is controlled using a Cartesian impedance control [142] with high stiffness gains (2000 N/m for the position and 200 Nm/rad for the orientation). Graduate students in robotics and automation participated to the experiments as teachers. The parameters used in this setup are listed in Table 5.3.

Pouring a drink

In the first experiment, the robot is tough how to pour water in a cup. The pouring task consists of two subtasks: *take-water* and *pour-water* (see Figure 5.18). During the teaching process, the teacher guides the robot towards the task execution, providing speech commands (*open*/*close*) to control the gripper. Figure 5.18 illustrates the WM state after a task demonstration (end of a teaching session). Nine atomic actions are generated at the end of the demonstration, and linked to the associated subtasks. These new elements are also associated with pre- and post-conditions, and with activation values. As detailed in Algorithm 4, these generated elements are also stored





(a) (b)

Figure 5.17.: Experimental setups used to test the framework for structured tasks learning and execution. (a) The setup used to acquire and reproduce uni-manual tasks. (b) The setup used to acquire and reproduce dual-arm tasks.

Table 5.3.: Parameters used in the experimental evaluation.

Robot Manager		
parameter	meaning	value
r	radius of the proximity area	0.12 m
$\boldsymbol{K} = \mathrm{diag}(k_1,..,k_6)$	DMP stiffness gains	$k_i = 70.0$
$\boldsymbol{D} = \mathrm{diag}(d_1,..,d_6)$	DMP damping gains	$d_i = 2\sqrt{k_i}$

Attentional System		
parameter	meaning	value
λ^{max}	max behavior period	1 s
λ^{min}	min behavior period	0.1 s
r^{max}	max object distance	2 m
r^{min}	min object distance	0 m
k	magnitude increment	1

in the LTM to be re-used in future scenarios. Once learned, the task can be executed. In order to execute the task, the attentional system first selects the subtask *take-water*, which is enabled when the robot has no object in its gripper (*hand.free*). The actions linked to the same subtask are executed in the order shown during the demonstration. For example, in order to perform the *take-water* subtask, the robot executes *foa1(water)*, *noa2(water)*, and then *gripper(close)*.

Teaching and execution times, measured over ten repetitions of the task, are used to quantitatively evaluate the effectiveness of the presented approach. Moreover, in order to show the robustness of the approach with respect to the initial conditions, ten repetitions of the task are performed with the bottle placed at random positions, measuring the *success rate* as the number of correct executions over the total executions. A trial is considered successful if the robot grasps the bottle and pours the water within the cup.

As shown in Table 5.4, teaching this relatively complex task requires approximately 50 s. More-

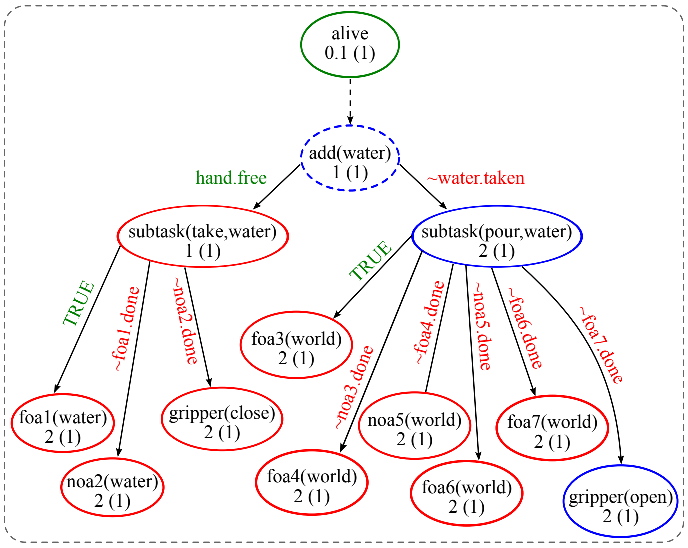


Figure 5.18.: The WM state after the pouring task demonstration. Nine generated segments are linked to the associated subtasks. Green are active and red inactive nodes. Blue are the last learned nodes.

over, the task was successfully executed in all the ten trials (success rate equal to 1). These results show that the framework allows to transfer novel skills to a robotic device in a quite fast, natural, and effective manner. Notice also that the execution time for the pouring task (77.5 s, on average) are slightly longer than the time needed to demonstrate the task (50.4 s, on average). This slower execution does not depend on the attentional system, which can monitor and select the robotic actions on-line. Instead, it mainly depends on the convergence time of the dynamical systems used to generate motor commands (see Section 3.1). A possible way to reduce the execution time is to perform each action at a predefined speed ε, i.e., by generating a velocity command with $\dot{\boldsymbol{x}} = \varepsilon \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$ instead of (3.3a).

Table 5.4.: Results for ten repetitions of the pouring a drink task.

Teaching Time [s] (mean ± std)			Teaching Time [s] (mean ± std)			Success Rate		
take-water	*pour-water*	*add(water)*	*take-water*	*pour-water*	*add(water)*	*take-water*	*pour-water*	*add(water)*
20.1 ± 1.2	30.3 ± 0.8	50.4 ± 2.0	31.0 ± 1.5	46.5 ± 0.8	77.5 ± 2.3	1	1	1

Prepare coffee - Task learning and autonomous execution

This experiment shows how a more complex structured task is learned and executed using the described framework. The task consists of preparing a coffee. In order to fulfill the task, the robot has to: *i)* pour the water in the cup, *ii)* add the coffee powder, and *iii)* mix water and coffee powder with a spoon. Before learning, the WM only contains the three subtasks *add(water)*, *add(coffee)* and *use(spoon)* without any link to atomic actions, as illustrated in Figure 5.19. Atomic actions are automatically added during the kinesthetic teaching and then used to reproduce the task. Note that the order of execution of *add(water)* and *add(coffee)* is not relevant for task learning and execution, therefore, they are both enabled when the task starts. In this case, task selection only depends on the attentional regulations. Figure 5.20 shows teaching and execution snapshots of *add(water)*, *add(coffee)*, and *use(spoon)*, each associated with the WM state obtained at the end

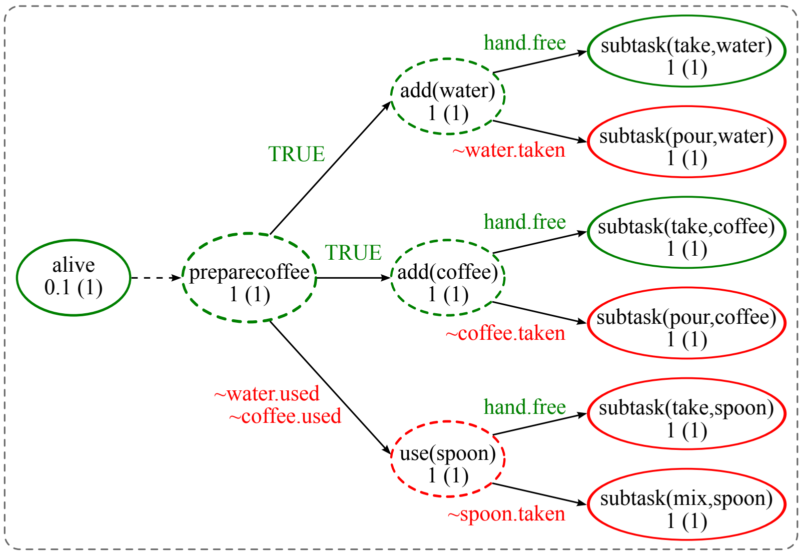


Figure 5.19.: The initial WM before learning how to prepare a coffee. The task *preparecoffee* has three child nodes, namely *add(water)*, *add(coffe)*, and *use(spoon)*. *add(water)* and *add(coffe)* can be executed in any order (*true* releaser), while *use(spoon)* requires that both the water and the coffee powder are added. Initially, both *subtask(take,water)* and *subtask(take,coffee)* are enabled (green), hence they compete for the initial segments.

of a learning session. Here, the user can directly teach the overall *prepare coffee* task and then execute it, otherwise the task can be step by step demonstrated and executed.

Similarly to the previous experiment, teaching and training time are measured, as well as, the success rate over ten task repetitions (with objects randomly placed). Results in Table 5.5 show that, on average, teaching the prepare coffee task takes less than 3 minutes, while executing the task takes about 3.7 minutes. Analogously to the previous experiment, the longer execution time mainly depends on the convergence time of the dynamical systems. Table 5.5 also shows training and execution times for each subtask. Looking at these results, it is possible to notice that the time to grasp an object is almost independent on the particular item. This is because, in the considered setup, objects are relatively close and they are grasped in a similar manner. It is also evident that *take-spoon* takes always longer than other take actions. The reason is that the subtask *use(spoon)* is always executed at the end, and the robot has to cover a bigger distance to reach the spoon. Moreover, Table 5.5 shows that the task execution has less variability than the teaching. This means that, despite the user introduces some variability across different demonstrations, the task execution time is relatively constrained. Several actions of the learned task are, in fact, linear point-to-point motions which are executed in similar times across different repetitions.

Also in this case, the task success rate is quite high (0.9) and only one failure occurs over ten trials. In the failed trial, the robot did not grasp the coffee jar sufficiently close to its center of mass, probably due to an error in the tracking system. Being the jar turned, the robot failed to add the coffee in the cup. Notice that the current implementation exploits a simple grasping strategy, i.e. reach the object and close the gripper. A possible way to increase the robustness of the system is to use a multi-fingered robotic hand and perform a power grasp [129], or to exploit tactile sensing in order to detect and avoid the slipping [41].

Table 5.5.: Results for ten repetitions of the prepare coffee task.

Teaching Time [s] (mean ± std)									
take-water	*pour-water*	*add(water)*	*take-coffee*	*pour-coffee*	*add(coffee)*	*take-spoon*	*mix-spoon*	*use(spoon)*	*prepareCoffee*
20.0 ± 2.3	30.1 ± 1.4	50.1 ± 3.7	19.8 ± 2.6	37.2 ± 1.9	57.0 ± 4.5	21.3 ± 2.7	37.3 ± 1.9	58.6 ± 4.6	165.7 ± 12.8

Execution Time [s] (mean ± std)									
take-water	*pour-water*	*add(water)*	*take-coffee*	*pour-coffee*	*add(coffee)*	*take-spoon*	*mix-spoon*	*use(spoon)*	*prepareCoffee*
28.4 ± 0.5	42.7 ± 0.3	71.1 ± 0.8	27.9 ± 0.7	47.7 ± 0.3	75.6 ± 1.0	29.8 ± 0.7	48.0 ± 0.4	77.8 ± 1.1	224.5 ± 2.9

Success Rate									
take-water	*pour-water*	*add(water)*	*take-coffee*	*pour-coffee*	*add(coffee)*	*take-spoon*	*mix-spoon*	*use(spoon)*	*prepareCoffee*
1	1	1	1	0.9	0.9	1	1	1	0.9

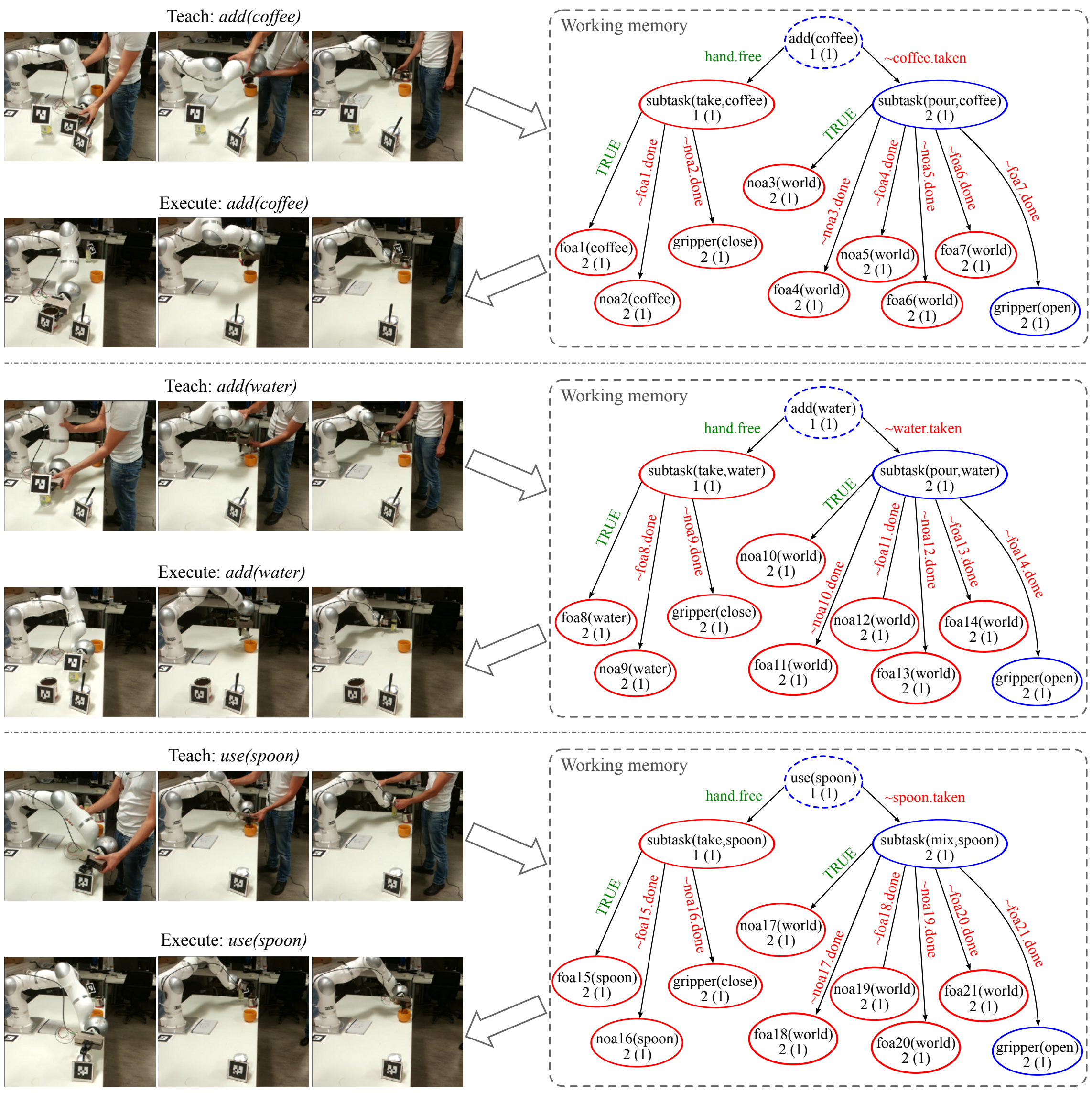


Figure 5.20.: The robot learns how to prepare a coffee. (Left panels) Snapshots of the kinesthetic teaching and autonomous task execution. (Right panels) Actions are automatically attached to the subtasks in the WM and used to reproduce the task.

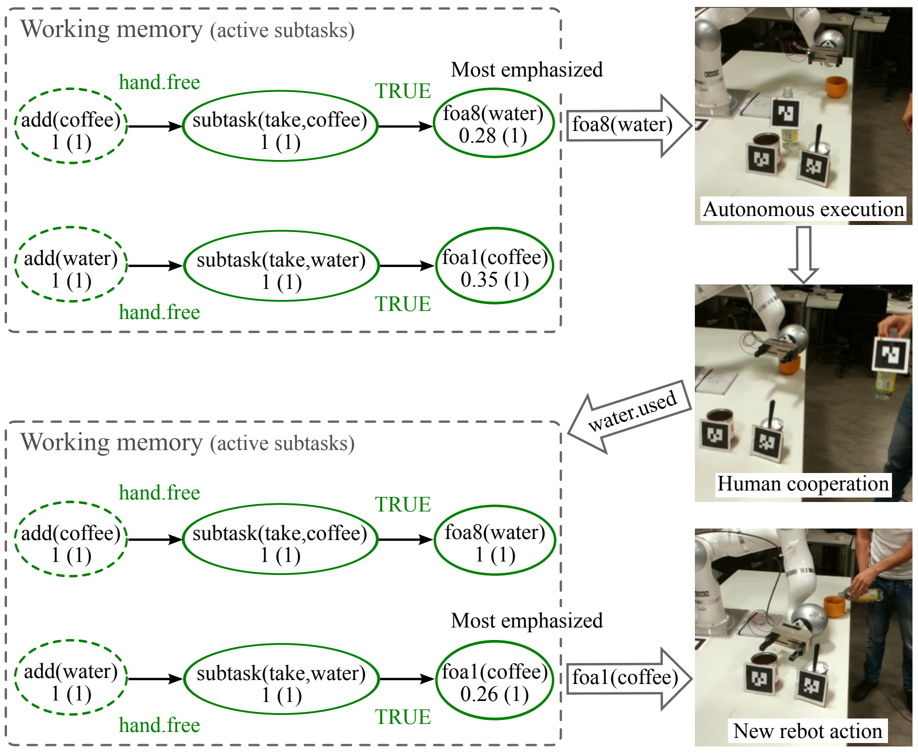


Figure 5.21.: Cooperative execution of the prepare coffee task. The user takes the bottle and pours the water while the robot is approaching the bottle. Notice that, before the human intervention the most emphasized action is *foa8(water)*. On the other hand, when the human performs the action, the robotic task execution is on-line adapted: the most emphasized action segment becomes *foa1(coffee)* and the robot takes the coffee jar.

Prepare coffee - Cooperative task execution

The presented framework permits a cooperative execution of the learned tasks. As a proof of concepts, this experiment considers the coffee task described in the previous experiment and two cooperative scenarios. In the first case, the human helps the robot to fulfill the task by adding the water himself. To show the on-line capabilities of the attentional system, the user intentionally takes the bottle while the robot is approaching it and pours the water himself. In the meanwhile, the robot was executing the *foa8(water)* action in Figure 5.21. Hence, the attentional system has to rapidly adapt task execution with respect to the human behavior. Since the water is not anymore available in the scene, the *add(water)* behavior becomes less attractive for the robot that starts to execute the *add(coffee)* (which is available and enabled in the WM). At the same time, the system can monitor the human behavior and assign the *add(water)* execution to the human. For the sake of simplicity, the above assignment is explicitly communicated by the human through a speech command. However, more complex activity recognition methods can be exploited for the same purpose [29]. The cooperative task is executed ten times, obtaining an execution time of 149.8 ± 3.5 s. A comparison with the autonomous execution time in Table 5.5 allows us to conclude that the cooperative execution is beneficial in terms of execution time. In particular, it is interesting to notice that the time needed for task replanning does not have a significant impact on the total execution time.

Human–robot cooperation can be also exploited to overcome robot limitations, as illustrated in Figure 5.22(a). In the considered case, the robot is pouring the water with the bottle closed.

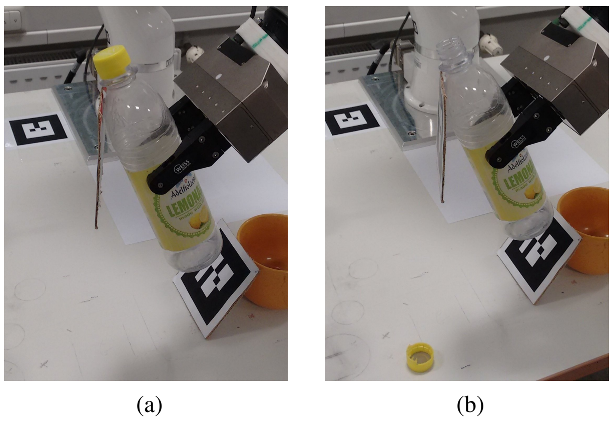


Figure 5.22.: Human–robot cooperation is exploited to overcome robot limitations. (a) The bottle is closed and the robot will fail to pour the water. (b) Once the user has suspended the task and removed the cap, the robot can correctly execute it.

There is no chance for a single robotic manipulator to fulfill this task. Indeed, even with a more sophisticated perception system able to recognize the cap, a single manipulator could not open the bottle and the pouring task would fail. In this case, the human intervention is essential to fulfill the task. Hence, during the execution phase, the user temporary suspends the task execution via the speech command *stop*, opens the bottle (see Figure 5.22(b)), and restarts the execution using the speech command *repeat*. Alternatively, it is possible to explicitly introduce in the task structure a subtask *open(Obj)* which is directly assigned to the human manipulation and left unlinked for the robot execution. In this way, the robot waits for the human help or guidance in order to execute the task. During the teaching phase, this subtask can be executed by the human (under the attentional supervision), while the rest of the task can be again demonstrated through kinesthetic teaching. It is worth considering that, in a cooperative scenario, the operator can teach motion primitives which favor the human interventions. For example, when the next subtask is a human manipulation (e.g. *open(water)*), the robot should provide the object in a comfortable position for the operator. This subtask can be also demonstrated in the learning phase taking into account the human intervention.

Prepare tea - Task re-usage

This experiment shows that the acquired knowledge can be re-used to speed-up the acquisition of novel tasks. The task consists of preparing a tea, where the robot has to pour the water in the cup and add a tea bag. The *add(water)* subtask is the same subtask used to prepare the coffee and can be re-used in this novel scenario. In other words, the already learned subtask can be loaded from the long term memory and instantiated in the working memory, while the user has only to teach how to add the tea bag (see Figure 5.23). Once allocated in the WM, all the enabled and linked subtasks (e.g. *add(water)*) can be executed until the open subtask (*add(tea)*) is selected. In this case the robot needs the human demonstration to learn how to complete the overall task. In order to assess the effectiveness of task re-usage, ten teaching sessions are performed: in five runs the teacher has to demonstrate the entire task, while in the remaining five runs the operator only teaches the missing *add(tea)* subtask. In the second case, the robot waits for the human assistance, being not able to execute the subtask. As reported in Table 5.6, task re-usage is effective and reduces the teaching time of about 53% in the considered prepare tea task.

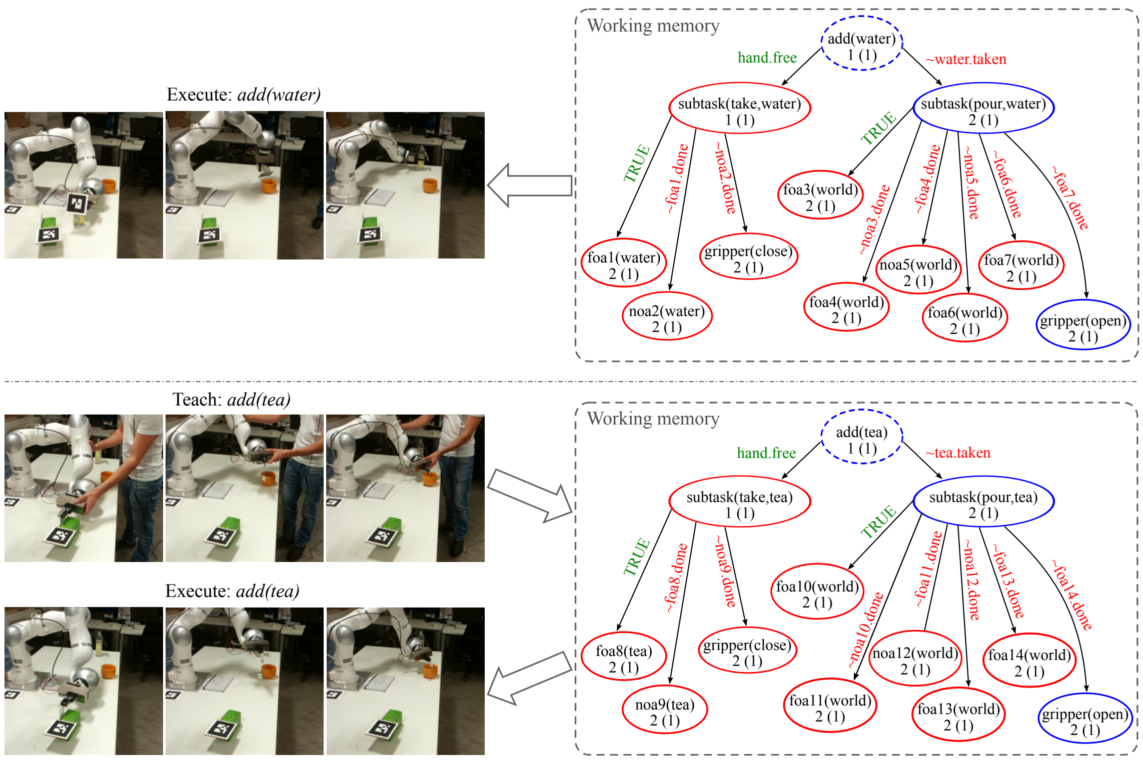


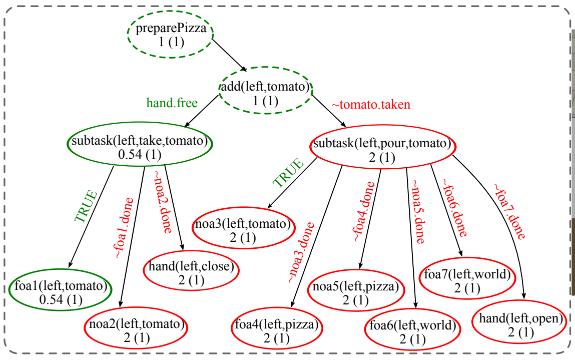
Figure 5.23.: The robot learns how to prepare a tea. (Top-left) The *add(water)* subtask has been already demonstrated for the *prepare coffee task* and can be reused in the *prepare tea task*. (Top-right) The WM state after the *add(water)* execution. (Bottom-left) The human can demonstrate the novel subtask through kinesthetic teaching, then the robot can autonomously execute the rest of the task (*add(tea)*). (Bottom-right) The WM state after the *add(tea)* demonstration.

Table 5.6.: Results for ten training trials of the prepare tea task. The symbol "–" indicates an already learned subtask.

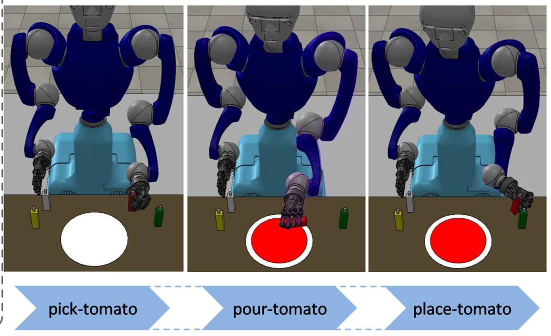
Teaching Time [s] (mean ± std)			Task re-usage
add(water)	*add(tea)*	*prepareTea*	(yes / no)
50.1 ± 1.9	34.1 ± 1.0	84.2 ± 2.9	no
–	35.6 ± 1.4	**35.6 ± 1.4**	yes

5.2.4. Dual-arm structured tasks

This section illustrates how the described framework can be effectively applied to learn and execute dual-arm structured tasks. To this end, the structured task of topping a pizza is considered. As shown in Figure 5.17(b) (left), the human teacher executes the task in front of a table where a set of real objects are disposed, namely a *pizza* and four bottles containing *tomato*, *oil*, *cheese*, and *basil*. The environment is reproduced in simulation (Figure 5.17(b), right), where the human operator is replaced with the simulated humanoid robot RoDyMan. As detailed in Section 4.2, a motion capture suite is employed for the human motion recording and imitation. Recall that the framework is able to learn and reproduce dual-arm tasks, where the motion of the two arms is



(a) Representation in WM of the learned subtask *add(tomato)* for the left arm.



(b) Execution of the *add(tomato)* task

Figure 5.24.: Execution of the *add(tomato)* task performed by the left arm. (a) The learned hierarchical structure contains three NOA, four FOA, and two hand commands. (b) The simulated robot reproduces the task exploiting the learned hierarchical structure.

not necessary synchronized (see Section 4.3.4). In other words, the arms are considered as two separate manipulators which are both able to perform the pizza topping task. The human teacher demonstrates the entire task twice, once using only the right arm and once using only the left arm. This is needed to make the two arms of RoDyMan able to independently execute the task. Both the training sessions last less then 4 minutes. A subset of the learned actions is shown in Figure 5.24. In particular, the figure shows that the task *add(tomato)*, learned by the left arm, is associated with 7 actions (3 NOA and 4 FOA) and 2 hand commands (*hand(left,close)* and *hand(left,open)*).

To evaluate the typical performance of the framework during the execution of the learned task, the pizza topping task is repeated 10 times by randomly changing the position of the ingredients on the table. It is worth noticing that no collision avoidance mechanisms are exploited during the task repetitions. Hence, in order to prevent self-collisions, the robot's workspace is split in three parts, namely left, right, and central areas. The left (right) area contains ingredients close to the left (right) arm. Therefore, the objects in the left and right areas can be manipulated with only one arm (left and right arm respectively). The central area contains the *pizza* and it is shared among the two arms.

Five task repetitions are initially performed using only one arm per time, with the ingredients randomly positioned inside the left (3 times) or right (2 times) areas. Execution times and success rates are reported in Table 5.7 (uni-manual row). Repetitions performed with one arm serve as baseline to assess the effectiveness of the dual-arm execution. Therefore, five more repetitions are performed by executing the task with both arms. In this case, the objects are randomly positioned in all areas and the robot exploits the tasks learned by the right and left arms in order to prepare the pizza. Snapshots of the dual-arm task execution are shown in Figure 5.25. It is worth noticing that the attentional system prevents the arms to enter simultaneously the central area, avoiding self-collisions between the arms. The central area, in fact, is treated as a shared resource with a mutually exclusive access. The arms compete to obtain the exclusive access to the central area. To regulate this conflict, a winner-take-all strategy is exploited where the resource (central area) is assigned to the arm that reaches first the boundary of the central area. Hence, a first come first served mechanism is exploited in this work, but other choices are possible. Table 5.7 illustrates the obtained results. All the 10 executions are successful. In particular, in the dual-arm case, the task is successfully executed without self-collisions. Moreover, performing the task with both arms improves the time performances by 50.8 seconds (18.04%). This results confirms that the

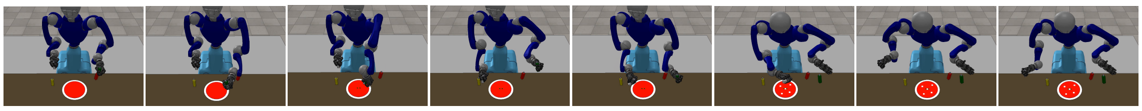


Figure 5.25.: Execution of the *add(cheese)* and the *add(basil)* tasks. The left and right arms cooperatively execute the task.

Table 5.7.: Results for ten repetitions of the pizza task.

	Execution Time [s] (mean $\pm$ std)	Success Rate
Dual-arm	281.6 ± 13.83	1
Uni-manual	332.4 ± 28.05	1

presented framework is able to efficiently combine the single-handed learned tasks, significantly reducing the overall task execution time.

5.3. Summary and conclusion

This chapter presented approaches to execute learned tasks on real robots. The first section exploited a hierarchical architecture for reactive planning of robotic skills. The low-level of the architecture generates the robot trajectory by exploiting stable dynamical systems (see Section 3.1), while taking into account the human safety and the correct execution of the robotic task. Regarding the human safety, the robot's velocity is automatically scaled down to a safe value in case of close interaction with the human. Moreover, possible collisions with unforeseen obstacles (including human beings) are prevented by exploiting the so-called dynamical system modulation, which generates collision-free paths without modifying the equilibria of the modulated system. The low-level has to match real-time requirements, being typical robot control frequencies in the range $[0.1, 1]$ kHz. Time constraints are satisfied by exploiting the real-time capabilities of the presented fDED (fast distance evaluation in the depth space) approach; an highly parallelizable algorithm that computes robot–obstacle distances in about 1 ms. The high-level of the architecture tracks eventual humans in the scene, and it leverages invariant motion representations (see Section 3.2) to robustly recognize human gestures. The recognized gestures are then transformed into commands for the robot, by exploiting dynamical systems properties to replan the current task on-line and generate a feasible robot's trajectory. The approach is evaluated in a realistic, dynamic scenario, showing promising results. In particular, dynamical systems properties and real-time distance computation effectively increase the human safety by reducing the robot's velocity in case of close interaction with the human, and by letting the robot avoiding possible collisions. Compared to geometric planners, the presented approach generates suboptimal paths. Geometric planners, in fact, consider human and robot kinematics, as well as objects in the scene, in order to search for the optimal (minimum-cost) path. Hence, geometric planners are able to find globally optimal paths, but they have high computational costs, which limits their applicability in dynamic environments. On the contrary, the trajectories generated with the presented approach are not necessarily optimal in terms of traveled distance or human safety, but the desired path is computed and replanned in real-time, allowing a more fluent human-robot interaction.

The second section focused on the execution of structured tasks, by exploiting the framework used in Section 4.3 for intuitive task transfer from a human teacher to a robotic device. The framework consists of a robot manager that handles the low-level aspects of the human–robot interaction, like the control modality switching, the segmentation of human demonstrations, and the generation of robot's trajectories, and of a supervisory attentional system that continuously monitors human and robot activities during both task learning and execution. The attentional system exploits contextual information, like robot–object distances and explicit human commands, to generate, and eventually replan, structured task plans. The framework has been evaluated considering robots operating in typical kitchen scenarios. In the first scenario, a robotic manipulator is used to prepare a drink (coffee or tea), autonomously or in cooperation with a human co-worker. In the second, simulated scenario, the humanoid robot RoDyMan executes the task of topping a pizza with several ingredients. Obtained results show that the system allows the robot to quickly learn and robustly execute typical structured activities that involve the manipulation of several objects. Moreover, experiments have shown how the attentional supervision of both the user and the robot activities enables cooperative execution of the learned tasks with an associated reduction of the execution time. Finally, the prepare tea task is used to show how an already learned task/subtask can be reused in a novel task, effectively enabling the acquisition of incrementally complex capabilities.

The approaches presented in this chapter have some limitations that can be overcome in a future research. First, the robot manager does not consider any human-aware replanning strategy. To this end, the hierarchical architecture used for on-line reshaping of robotic skills can be integrated in the robot manager and used to reshape the learned motion primitives. Moreover, instead of simply transforming human gestures into commands for the robot, the attentional system can be exploited in more complex human–robot interaction scenarios along with more sophisticated attentional cueing mechanisms. The described integration and extension of the robot manager will allow the safe execution of structured tasks. Another limitation of the presented framework is that it does not allow incremental refinement of learned tasks. To this end, one can incorporate in the robot manager further control modalities which permit physical human–robot interaction during the task execution. Examples of such control modalities are the interaction control presented in Section 4.1 and the variable impedance control presented in [91]. Another interesting research direction is to extend the framework for structured tasks execution to learn and execute force patterns, enabling the robot to execute dynamic tasks. To this end, the robot manager has to be extended in order to segment both force and kinematic trajectories, as well as to represent and execute motion primitives that require explicit force control. Finally, in this work the abstract task, like prepare coffee, is assigned a priori by an expert user and loaded from the long term memory. An interesting improvement is to estimate the task to execute from the current scene, enabling the robot to automatically load in its working memory the tasks/subtasks more appropriate for the context at hand.

CHAPTER 6

Conclusion

Robots operating in human populated environment are asked to efficiently acquire incrementally complex tasks and to fruitfully execute a variety of activities autonomously or in cooperation with other agents, including human co-workers. In order to integrate robotic solutions in social applications, several aspects of the task acquisition process have to be considered and fused together into a framework that allows a fluent acquisition of novel tasks. This thesis work represents an attempt to build such a framework for intuitive tasks acquisition, permitting a quick task learning and a flexible task execution in realistic environments. The work focused on the principal aspects of the task acquisition process, namely the intuitive task learning, the efficient task representation, and the effective task execution. These three aspects represent the building blocks, or the main components, of the framework developed in this work.

6.1. Task representation

Representing the task knowledge in a proper manner is of fundamental importance in order to properly acquire novel tasks. As discussed in Chapter 3, in this work the task knowledge is represented at two different levels of representation. Elementary movements, also called motion primitives, are represented in a compact form which allows real-time motion generation and generalization, and reduces memory requirements. Stable dynamical systems are identified as the better-suited representation of point-to-point movements. In particular, the thesis presents an effective approach to encode motion demonstrations into a stable system, represented by Gaussian mixture regression. The system is learned from demonstrations and stabilized at run time to guarantee global convergence towards a unique target. The comparison carried out in Section 3.1.4 shows that the presented approach is slightly less accurate than state-of-the-art approaches but significantly faster, offering a good compromise between the accurate reproduction of the demonstrated movements and the training time. Hence, the presented approach is a suitable choice to represent multiple demonstrations of the same movement into a compact form, while other approaches in the literature, like the dynamic movement primitives [65], are better-suited when single demonstrations are given.

The limitation of Cartesian trajectories as motion descriptors are also investigated in this work. Cartesian based descriptors, in fact, are affected by several motion variations, like the point of view and the motion execution style. These limitations are overcome by transforming Cartesian trajectories into a different space with known invariance properties. Section 3.2.2 presented the Denavit–Hartenberg inspired Bidirectional (DHB) representation; a minimal (six values), bidirectional, invariant to affine transformations, linear, angular, and temporal scalings, and numerically robust representation of rigid body motions. The adoption of such an invariant representation is beneficial in both gesture recognition and reproduction problems. The invariance properties are helpful to cope with different observation points, as well as with different executions styles and kinematics of various subjects, as shown in several gesture recognition experiments carried out in Section 3.2.3. Bidirectional invariant representations, i.e. representations which allow to reconstruct a Cartesian trajectory from an invariant one, are useful in motion generation problems. As shown in Section 3.2.4, several Cartesian trajectory instances can be generated from a unique invariant representation, increasing the generalization capabilities of the motion generation algorithm. The limitation of invariant representations is that they are not compact, since they have almost the same dimensionality of Cartesian trajectories. Representing invariant trajectories in a compact from, like a stable dynamical system, will definitely increase the usability of such representations in robotic applications.

The mentioned approaches for representing robotic skills work for basic motion units like point-to-point motions. However, the framework presented in this work is capable of intuitively acquiring complex robotics tasks, in particular structured tasks. A structured task has been defined as a task consisting of more than one basic action with associated execution constraints. Moreover, a structured task can be hierarchically decomposed into several levels. As detailed in Section 3.3, a rooted tree is the better-suited topology to represent a structured task. In particular, this work adopt rooted trees with three or four levels where the root is simply a label used to distinguish the task at hand, the so-called abstract task. The root node is connected to two or three more levels. Going down the tree the levels become incrementally more specific until the leaves level is reached. The leaves, in fact, contain the basic action that the robot has to perform in order to execute the task. Each node in the tree is associated with logical pre- and post-conditions and a weight or emphasis. These informations are exploited at run time to decide the next action that the robot has to execute in order to complete a certain task.

6.2. Task learning from human demonstrations

Human beings are able to easily acquire novel tasks, for example by observing other humans while performing the task. Learning from human observation is an intuitive and quick way of transferring task knowledge from a human teacher to a learning agent, and it is widely used in this work to learn novel robotic tasks. Kinesthetic teaching, i.e. physically guiding the robot towards the task execution, and imitation learning, i.e. tracking a user during the task execution and mapping the motion to a robot, are the two solutions exploited in this work for intuitive skills transfer. Advantages and limitations of the two solutions are discussed at the beginning of Chapter 4, showing that imitation learning is better suited when human-like (humanoid) robots are used. Indeed, the main drawback of imitation learning is that human movements have to be mapped to a humanoid robot, which usually has a different kinematic structure compared to the human beings. This problem is known as the motion retargetting or the correspondence problem. In this work, the motion retargetting problem is solved with the approach presented in Section 4.2, that leverages an inverse kinematics algorithm and simple geometric considerations.

Kinesthetic teaching is better suited for robots with less degrees-of-freedom than a humanoid robot. This is the case of robotic manipulators which usually have six or seven joints. Depending on the task, robotic manipulators may have a variable number of redundant degrees-of-freedom (DoF). For instance, a robot with seven joints has at least one redundant DoF, this independently from the motion to execute. When a robot is redundant, the extra DoF can be exploited to generate the so-called null-space movements, which are motions of the robot body that do not affect the pose of the end-effector. This work presents an approach for incremental kinesthetic teaching of end-effector and null-space tasks. The approach leverages an incremental learning algorithm, external forces estimation, and a customized kinematic control to permit a human teacher to correct the robotic task execution, by physically interacting with any point on the robot body. Several experiments in Section 4.1.3 show that the presented approach represents a valid solution for intuitive and incremental transfer of end-effector and null-space movements.

Kinesthetic teaching and imitation learning are usually exploited to transfer basic skills from a human teacher to a robotic learner. This thesis extends the standard approaches presenting a framework that combines motion segmentation, learning from demonstrations, and attention supervision to realize an intuitive transfer of structured tasks. In particular, a task demonstration is firstly segmented into basic motion units. Each segmented motion is automatically labeled and associated to an object to manipulate, by exploiting the segmentation strategy described in Section 4.3.1. Segmented data are then encoded into dynamic movement primitives, which allow the generation of motor commands for the robot. At a higher level, an attentional supervisory system monitors the human and the robot activities in order to relate the generated action labels and execution constraints to the task structure. As detailed in Section 4.3.2, the attentional supervisory system exploits cognitive control mechanisms to determine the most emphasized robot's behavior, i.e. the most active subtask in the tree, and it assigns the generated label to this behavior. The described framework permits to learn both a dynamical system based and a symbolic representation of structured tasks. Hence, after the learning session, the robot has fully acquired the novel task and it is able to execute the demonstrated structured task, without any further human intervention or data post-processing.

6.3. Task generation and execution

Task generation and execution is the last aspect of the task acquisition process considered in this work. Indeed, a robot has not fully acquired a novel task unless it is able to reproduce the task. Moreover, in everyday scenarios, robots are asked to modify their behavior to cope with sudden and unexpected changes in the environment, as well as to consider human intentions and needs. As for the task representation and learning, the approaches for the task generation and execution presented in Chapter 5 follow a hierarchical approach. In particular, the attentional supervisory system exploits the learned task tree and the current context to select the next action that the robot has to execute. The attentional system simply generates a set of labels used to identify the action to execute. A lower layer is then responsible to generate motor commands for the robot. This layer leverages dynamical systems to generate the motion trajectories and to online reshape the generated trajectory in case of changes in the robot's workspace.

More in detail, the lower layer takes as input the action (label) to execute. The action is selected from a database of movement primitives, encoded into stable dynamical systems. Motion primitives can be learned from demonstration or provided by an expert user. The dynamical system representation allows to modify online the robot's trajectory, for example to avoid possible collisions or increase the human safety. As detailed in Section 5.1, the velocity generated from a dynamical system can be rescaled in case of close interaction with the human. This reduces the severity of possible injuries due to an accidental collision while preserving the correct task execution, since the rescaled system is still guaranteed to converge to the goal. In case of unforeseen objects present in the scene, the robot's velocity is modified to prevent possible collisions, again without modifying the equilibrium points of the dynamical system. To this end, a suitable modulation matrix is computed given the minimum robot–obstacle distance and the normal vector connecting the points of minimum distance on the object and on the robot body. The so-called dynamical system modulation approach has been tested in simulation and experiments on real robots, in both static and dynamic environments, allowing the robot to prevent possible collisions without affecting the correct task execution.

The reactive reshaping mechanism requires a continuous monitoring of the robot's workspace. RGB-D camera sensors are widely used in this work to monitor the scene and to track eventual human co-workers. Apart from tracking the human, sensory data are also used to compute the distance between the robot and eventual objects in the scene. In particular, Section 5.1.2 present an approach capable to compute robot–obstacles distances in real-time, i.e. within 1 ms. The approach takes as input the depth image from an RGB-D camera and it directly computes the distance in the depth space. In this work, the robot body is represented with a triangular mesh model. In contrast to state-of-the-art approaches, which usually approximate the robot with geometric primitives like spheres or cylinders, the mesh model represents the robot more accurately, resulting in a more accurate distance evaluation. The drawback is that a mesh models has usually thousands of points, which makes the distance evaluation computationally expensive. In order to work in real-time, the approach in Section 5.1.2 creates a lattice of robot and object points, containing significantly less points than the original depth image, and iteratively refines the distance computation to increase the accuracy. All the operations are highly parallelizable and executed on the graphic processing unit. The algorithm has been tested in a variety of realistic conditions, showing promising results in terms of accurate distance evaluation and reduced computation time. Moreover, the approach has been effectively combined with the reactive planning approach in Section 5.1.1. Compared with slower distance evaluation techniques, the presented approach is more effective in avoiding collisions with moving obstacles (including humans). Hence, the real-time capabilities of the presented technique effectively increase the human safety in human–robot interaction scenarios.

The described reactive reshaping level takes as input the next action to execute and generate a feasible robot's trajectory. This is highly compatible with the attentional supervision layer, used to generate the task plan. The attentional supervisory system, in fact, commands to the robot the next action to execute considering the current context and the task at hand. The commanded action is nothing more than a unique label, that the lower layer uses to find the proper action from the database of motion primitive and to generate a feasible robot's trajectory. Hence, in this work, the task planning problem is separated into two steps. An high level attentional mechanism generates a symbolic plan, i.e. it selects and commands to the robot the next action to execute, while a low level "robot manager" effectively generates the motor commands. The separation into symbolic planning and motion generation allows a quick task replanning and a fluent human–robot interaction. As already mentioned, the robot manager is able to modify the robot's trajectory in real-time to avoid possible collisions and to ensure the human safety, without compromising the correct task execution.

The higher level, instead, takes care of more complex aspects of the interaction. As discussed in Section 5.2, the attentional system periodically expands the nodes in the task structure to select the most emphasized behavior and decide the next action to command. The selection process depends on the task structure (pre- and post-conditions of each action), the current state of the task (last executed action), contextual informations, and explicit human commands. Human commands directly affects the task execution. For example, speech commands are used to switch between teaching and execution phase, or to suspend the execution of a certain task. This work used only speech commands, but other choices (e.g., gestures) are possible. Contextual informations, like objects in the scene and robot–object distances, directly affects the emphasis weight of each node, favoring the execution of certain behaviors. The task learning and generation capabilities of the presented framework have been evaluated in typical kitchen scenarios. Obtained results show that the framework allows the robot to quickly learn and robustly execute typical structured activities that combine pick, place, and object manipulation actions. Moreover, experiments have shown that the attentional supervision of both user and robot activities enables cooperative task execution with an associated reduction of the execution time. Finally, experiments have illustrated how an already acquired task can be reused in a novel task, enabling the acquisition of incrementally complex abilities.

6.4. Future research directions

Limitations of the presented approaches have been discussed in the conclusion section of each chapter, together with possible improvements and future work. The aim of this section is to give a final overview of the research carried out in this thesis and to underline which are the possible research lines in the field of intuitive skills transfer.

The presented framework for intuitive task acquisition separates the task planning problem in two parts, where the attentional system generates a symbolic plan and the robot manager generates the motor commands. The two layers are conceptually separated, but they exchange information regarding the task execution. Interestingly, the lower level has to communicate to the higher level that the commanded action has been successfully executed. This mechanism is not fully exploited in the current version of the framework, since the robot manager is not able to understand that the execution of an action failed. Hence, an interesting further development is to integrate a real-time failure detection approach in the robot manager. Developing an effective and fast mechanism for abnormalities detection, in fact, is the first step to realize a system able to recover from eventual problems that may occur during the task execution. Failures occurred during the execution of an action can be detected and classified through visual and/or tactile data, by considering the failure detection as a binary classification problem (success or failure), where successful task demonstrations are considered as training data. The robot manager communicates eventual failures to the attentional supervision system. In case of failures, the attentional system can select the most appropriate recovery strategy from a database of motion primitives. Such a database is created from human observation or provided by an expert user. Selecting the recovery strategy from a database will significantly reduce the time needed to re-plan the task, making possible to realize an effective strategy for on-line task re-planning.

The work in this thesis was focused on realizing a framework for intuitive task transfer from human to robotic devices. Programming by demonstration has been identified as a key aspect of the intuitive transfer, giving the user the possibility to demonstrate a certain task instead of hand programming the robot behavior. However, programming by demonstration becomes challenging for robots with many degrees-of-freedom, as underlined by the user study in [162]. The solution exploited in this work to teach humanoid robots is to apply imitation learning and motion retargeting. An alternative and promising solution is to realize a framework for assisted skills transfer, where the robot uses already acquired knowledge to assist the human during the teaching. Hence, one has to move from the one-directional programming by demonstration paradigm, where the human teacher is the only owner of the task knowledge, to a bi-directional paradigm where the robot exploits previous knowledge to facilitate the acquisition of novel tasks.

Appendices

APPENDIX A

Materials and methods

A.1. Dynamical systems theory

A first-order, autonomous dynamical system (DS) is formally defined as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)), \tag{A.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the so-called *state* of the DS, $\dot{\boldsymbol{x}}(t) \in \mathbb{R}^n$ is the time derivative of $\boldsymbol{x}(t)$, and $\boldsymbol{f}(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is a continuous and continuously differentiable non-linear function. In robotics applications, the state $\boldsymbol{x}(t)$ usually represents the position of the robot (in joint or Cartesian space), and $\dot{\boldsymbol{x}}(t)$ its velocity [18, 76, 78, 117]. Note that, in order to simplify the notation, the time dependency of $\boldsymbol{x}$ is omitted in the rest of the chapter. A solution of (A.1), namely $\boldsymbol{\Phi}(\boldsymbol{x}_0, t) \in \mathbb{R}^n$, is called trajectory. Different initial conditions $\boldsymbol{x}_0$ generate different trajectories. A point $\boldsymbol{x}^*$ such that $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0} \in \mathbb{R}^n$ is an equilibrium point of the DS. An equilibrium is locally asymptotically stable (LAS) if $\lim_{t\to+\infty} \boldsymbol{\Phi}(\boldsymbol{x}_0, t) = \boldsymbol{x}^*, \forall \boldsymbol{x}_0 \in \mathcal{D} \subset \mathbb{R}^n$. If $\mathcal{D} = \mathbb{R}^n$, then $\boldsymbol{x}^*$ is globally asymptotically stable (GAS) and it is the only equilibrium of the DS. The stability of an equilibrium point can be proved by using results from Lyapunov or Contraction theory. Some useful results concerning the stability of the equilibrium points of non-linear dynamical system are summarized as follows.

A.1.1. Lyapunov theory

The stability of an equilibrium point can be studied by analyzing the behavior of a scalar function of the state variable $V(\boldsymbol{x}) \in \mathbb{R}$. As stated by the so-called *Lyapunov theorem* [146], an equilibrium $\boldsymbol{x}^*$ is LAS in a compact region $\mathcal{D} \subset \mathbb{R}^n$ if a Lyapunov function $V(\boldsymbol{x})$ exists that satisfies the following conditions:

$$V(\boldsymbol{x}) \geq 0,\ \forall \boldsymbol{x} \in \mathcal{D} \text{ and } V(\boldsymbol{x}) = 0 \Longleftrightarrow \boldsymbol{x} = \boldsymbol{x}^*, \tag{A.2a}$$

$$V(\boldsymbol{x}) \to \infty \text{ as } \|\boldsymbol{x}\| \to \infty \text{ (radially unbounded)}, \tag{A.2b}$$

$$\dot{V}(\boldsymbol{x}) \leq 0,\ \forall \boldsymbol{x} \in \mathcal{D} \text{ and } \dot{V}(\boldsymbol{x}) = 0 \Longleftrightarrow \boldsymbol{x} = \boldsymbol{x}^*. \tag{A.2c}$$

A function that satisfies conditions (A.2a)–(A.2c) is called a Lyapunov function. Moreover, the equilibrium point is GAS if conditions (A.2a)–(A.2c) are satisfied for all $\boldsymbol{x} \in \mathbb{R}^n$.

A.1.2. Contraction theory

Contraction theory [95] is an alternative approach to analyze the stability of non-linear dynamical systems in the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},t), \tag{A.3}$$

where $\boldsymbol{f}(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is a non-linear and, in general, time depending smooth function. Contraction theory is based on the idea that, if moving along a trajectory of the contracting system, the virtual (point-wise) distance to its neighboring trajectories exponentially decreases. This idea is formalized as follows. For the smooth DS (A.3), it holds that $\delta\dot{\boldsymbol{x}}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{x},t)\delta\boldsymbol{x}(t)$, where $\delta\boldsymbol{x}(t)$ is an infinitesimal virtual displacement between any two trajectories of (A.3). In general, one can consider a differential coordinate transformation of the virtual displacement $\delta\boldsymbol{x}(t)$ via an invertible (full rank) square matrix $\boldsymbol{\Theta}(\boldsymbol{x},t)$: $\delta\boldsymbol{z}(t) = \boldsymbol{\Theta}(\boldsymbol{x},t)\delta\boldsymbol{x}(t)$. The time derivative $\delta\dot{\boldsymbol{z}}$ can be written as

$$\delta\dot{\boldsymbol{z}}(t) = \left(\dot{\boldsymbol{\Theta}} + \boldsymbol{\Theta}\frac{\partial \boldsymbol{f}(\boldsymbol{x},t)}{\partial \boldsymbol{x}}\right)\boldsymbol{\Theta}^{-1}\delta\boldsymbol{z}(t) = \boldsymbol{F}(\boldsymbol{x},t)\delta\boldsymbol{z}(t), \tag{A.4}$$

where the dependencies of $\boldsymbol{\Theta}$ are omitted to simplify the notation. $\boldsymbol{F}(\boldsymbol{x},t)$ in (A.4) is the so-called generalized Jacobian. Considering the positive definite metric $\boldsymbol{M}(\boldsymbol{x},t) = \boldsymbol{\Theta}(\boldsymbol{x},t)^{\mathrm{T}}\boldsymbol{\Theta}(\boldsymbol{x},t)$, the squared point-wise distance between any two trajectories of (A.3) is defined as $\delta\boldsymbol{z}(t)^{\mathrm{T}}\delta\boldsymbol{z}(t) = \delta\boldsymbol{x}(t)^{\mathrm{T}}\boldsymbol{M}(\boldsymbol{x},t)\delta\boldsymbol{x}(t)$, and its time derivative is

$$\frac{d}{dt}\delta\boldsymbol{z}(t)^{\mathrm{T}}\delta\boldsymbol{z}(t) = 2\delta\boldsymbol{z}(t)^{\mathrm{T}}\frac{d}{dt}\boldsymbol{z}(t) = 2\delta\boldsymbol{z}(t)^{\mathrm{T}}\boldsymbol{F}(\boldsymbol{x},t)\delta\boldsymbol{z}(t), \tag{A.5}$$

where the second equality derives from (A.4). If $\boldsymbol{F}(\boldsymbol{x},t)$ is uniformly negative definite in a region $\mathcal{C} \subseteq \mathbb{R}^n$ (see *Definition 3* in Table A.1), then $\delta\boldsymbol{z}(t)^{\mathrm{T}}\delta\boldsymbol{z}(t)$, and consequently $\delta\boldsymbol{x}(t)^{\mathrm{T}}\delta\boldsymbol{x}(t)$, exponentially converge to zero. Therefore, all the trajectories of (A.3) converge exponentially to each other in the region $\mathcal{C} \subseteq \mathbb{R}^n$, and $\mathcal{C}$ is called a *contraction region* with respect to $\boldsymbol{M}(\boldsymbol{x},t)$. This result is stated by the following theorem.

Theorem 1. [1]*Consider the dynamical system (A.3) and a given trajectory $\hat{\boldsymbol{x}}(t)$, where $\hat{\boldsymbol{x}}(t)$ is a solution of (A.3). Consider also a ball $\mathcal{B}$ of constant radius with respect to the metric $\boldsymbol{M}(\boldsymbol{x},t) = \boldsymbol{\Theta}(\boldsymbol{x},t)^{\mathrm{T}}\boldsymbol{\Theta}(\boldsymbol{x},t)$, centered at $\hat{\boldsymbol{x}}(t)$. Any trajectory of (A.3), which starts in $\mathcal{B}$ and is contained at all times in a contraction region $\mathcal{C}$ with respect to $\boldsymbol{M}(\boldsymbol{x},t)$, remains in $\mathcal{B}$ and converges exponentially to $\hat{\boldsymbol{x}}(t)$.*
Furthermore, global exponential convergence to $\hat{\boldsymbol{x}}(t)$ is guaranteed if the whole state space is a contraction region with respect to the metric $\boldsymbol{M}(\boldsymbol{x},t)$. ■

When the conditions in *Theorem 1* hold, the trajectories of the DS converge towards an equilibrium trajectory, which can be either an equilibrium point or a periodic orbit. Regarding the convergence towards an equilibrium point, the following result holds.

Corollary 1. [1]*If a system is globally contracting and autonomous, all the trajectories converge exponentially towards a unique equilibrium point.* ■

Contraction of a system can be proven by analyzing the matrix measure of the generalized Jacobian. The matrix measures associated to Euclidean and l_1 norms are:

$$\mu_2(\boldsymbol{A}) = \max_i \left(\lambda_i\left\{\frac{\boldsymbol{A}+\boldsymbol{A}^{\mathrm{T}}}{2}\right\}\right), \tag{A.6}$$

$$\mu_1(\boldsymbol{A}) = \max_j \left(a_{jj} + \sum_{i\neq j}|a_{ij}|\right), \tag{A.7}$$

[1]Refer to [95, p. 2-3], [95, p. 8], [95, p. 10] and [159, p. 4] for the proofs of *Theorem 1-3* and *Corollary 1*.

Table A.1.: Definitions

Definition 1: Given a square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, the matrix $\boldsymbol{S} = \frac{1}{2}\left(\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{A}\right) \in \mathbb{R}^{n\times n}$ is the symmetric part of $\boldsymbol{A}$.

Definition 2: Let $||\boldsymbol{A}||_i$ be an induced matrix norm of $\boldsymbol{A}$ on $\mathbb{R}^{n\times n}$. Then the corresponding **matrix measure** is the function $\mu(\boldsymbol{A}) : \mathbb{R}^{n\times n} \to \mathbb{R}$ defined by $\mu(\boldsymbol{A}) = \lim_{\varepsilon\to 0^+} \frac{||\boldsymbol{I}+\varepsilon\boldsymbol{A}||_i - 1}{\varepsilon}$. Useful properties of $\mu(\boldsymbol{A})$ are:

1) $\mu(\alpha\boldsymbol{A}) = \alpha\mu(\boldsymbol{A}), \forall\boldsymbol{A} \in \mathbb{R}^{n\times n}, \alpha \geq 0$.

2) $\mu(\boldsymbol{A}+\boldsymbol{B}) \leq \mu(\boldsymbol{A}) + \mu(\boldsymbol{B}), \forall\boldsymbol{A},\boldsymbol{B} \in \mathbb{R}^{n\times n}$.

3) For any nonsingular matrix $\boldsymbol{N}$ and any vector norm $||\cdot||$ with induced matrix measure μ, $||\boldsymbol{N}\boldsymbol{x}||$ is a vector norm and its induced matrix measure μ_N is $\mu_N(\boldsymbol{A}) = \mu(\boldsymbol{N}\boldsymbol{A}\boldsymbol{N}^{-1})$.

Definition 3: Given the system equations $\dot{\boldsymbol{x}} = f(\boldsymbol{x},t)$, the Jacobian $\boldsymbol{F}(\boldsymbol{x},t) = \partial\boldsymbol{f}/\partial\boldsymbol{x}$ is uniformly negative definite if $\exists\beta > 0 | \forall\boldsymbol{x} \in \mathcal{C} \subseteq \mathbb{R}^n, \forall t \geq 0, \frac{1}{2}\left(\boldsymbol{F}(\boldsymbol{x},t)^{\mathrm{T}} + \boldsymbol{F}(\boldsymbol{x},t)\right) \leq -\beta\boldsymbol{I} < 0$.

where $\lambda_i\{\boldsymbol{B}\}$ in (A.6) is the i-th eigenvalue of $\boldsymbol{B}$. Recall that, unlike vector measures, matrix measures can be also negative. Note also that the statements $\mu_2(\boldsymbol{A}) < 0$ and *A is negative definite* are equivalent. In fact, the results in *Theorem 1* also hold for other matrix measures.

Theorem 2. [1]*The DS $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},t)$ is contracting if it exists an invertible matrix $\boldsymbol{\Theta}(\boldsymbol{x},t)$ such that $\boldsymbol{M}(\boldsymbol{x},t) = \boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{\Theta}$ is positive definite and the generalized Jacobian satisfies $\exists c > 0\ |\mu(\boldsymbol{F}(\boldsymbol{x},t)) \leq -c, \forall t \geq t_0$, where $\mu(\cdot)$ is a matrix measure associated to a vector norm in Euclidean space.* ■

A.1.3. Partial contraction theory

In general, the global uniform negative definiteness of the generalized Jacobian is not trivial to prove [159]. Indeed, Contraction is a conservative condition that many DS do not satisfy. A less conservative extension of Contraction theory, namely the Partial Contraction theory, is proposed in [159]. Partial Contraction theory applies to DS in the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{x},t). \tag{A.8}$$

For instance, the DS $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},t) + \boldsymbol{g}(\boldsymbol{x},t)$ and $\dot{\boldsymbol{x}} = a(x)\boldsymbol{f}(\boldsymbol{x},t)$ are in the form (A.8). Instead of guaranteeing that (A.8) is contracting, an *auxiliary* DS is considered, namely

$$\dot{\boldsymbol{y}} = f(\boldsymbol{y},\boldsymbol{x},t). \tag{A.9}$$

Theorem 3. [1]*Assume that the auxiliary system (A.9) is contracting with respect to $\boldsymbol{y}$. If a particular solution of the auxiliary y-system (A.9) verifies a smooth specific property, then all trajectories of the original x-system (A.8) verify this property exponentially. The original system is said to be partially contracting.* ■

The smooth specific property can be any relationship between the state variables of (A.8) and (A.9), e.g., an equilibrium point.

A.2. Robot control

A.2.1. Rigid Body Motion Representation

Rigid body motions are conveniently represented by attaching an orthogonal frame to the rigid body (body frame), and by describing the pose (position and orientation) of the body frame with respect to a fixed frame (world frame). In each time instant, the position of the rigid body is represented by the vector $\boldsymbol{p} \in \mathbb{R}^3$ connecting the origin of the body frame with the origin of the world frame. The axes of the body frame can be projected along the axes of the world frame by the means of the *direction cosines*. Hence, the orientation of the rigid body is described by collecting the direction cosines into a 3×3 rotation matrix $\boldsymbol{R}$. It is possible to show that a minimal representation of the orientation consists of 3 values [142]. A possible way to represent the orientation is to use the three angles representing the three elementary rotations, i.e. rotations around a single axis, which align the body frame with the world frame. These angles are called *Euler angles*. There are 27 possible combinations of Euler angles, which become 12 by preventing consecutive rotations about parallel axes. In this work, *roll-pitch-yaw* Euler angles [142] are used to control the orientation of the KUKA LWR [15].

Another minimal representation exploited in this work is the *rotation vector*. The rotation vector $\boldsymbol{r} = \theta\hat{\boldsymbol{r}} \in \mathbb{R}^3$ is computed from the rotation matrix $\boldsymbol{R}$ as

$$\theta = \arccos\left(\frac{\operatorname{trace}(\boldsymbol{R}) - 1}{2}\right), \quad \hat{\boldsymbol{r}} = \frac{1}{2\sin\theta}\begin{bmatrix} \boldsymbol{R}(3,2) - \boldsymbol{R}(2,3) \\ \boldsymbol{R}(1,3) - \boldsymbol{R}(3,1) \\ \boldsymbol{R}(2,1) - \boldsymbol{R}(1,2) \end{bmatrix},$$

where the trace$(\boldsymbol{A})$ is the sum of the diagonal elements of the matrix $\boldsymbol{A}$. The rotation matrix $\boldsymbol{R}$ is computed from $\boldsymbol{r}$ by means of the exponential map

$$\boldsymbol{R} = \exp(\boldsymbol{r}) = \boldsymbol{I} + \frac{\boldsymbol{S}(\boldsymbol{r})}{\theta}\sin(\theta) + \frac{\boldsymbol{S}^2(\boldsymbol{r})}{\theta^2}(1 - \cos(\theta)),$$

where the skew-symmetric matrix $\boldsymbol{S}(\boldsymbol{r}) \in \mathbb{R}^{3\times 3}$ is given by

$$\boldsymbol{S}(\boldsymbol{r}) = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}.$$

A.2.2. Kinematic control

The forward kinematics problem consists in finding the mapping between the joint space $S_q \in \mathbb{R}^m$ and the task (Cartesian) space $S_x \in \mathbb{R}^n$ of the robot. The inverse mapping from the task space $S_x \in \mathbb{R}^n$ to the joint space $S_q \in \mathbb{R}^m$ is the inverse kinematics problem. The forward kinematics problem always admits a unique solutions and it is usually solved by applying the Denavit-Hartenberg procedure [44]. Then the forward kinematics can be written as

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}), \tag{A.10}$$

where the kinematic function $\boldsymbol{f} : S_x \longrightarrow S_q$. The time dependency of $\boldsymbol{x}$ and $\boldsymbol{q}$ is omitted in (A.10) to simplify the notation. Note that, depending on the task space definition, the vector $\boldsymbol{x}$ may contain Cartesian positions, orientations, or both.

The inverse kinematics problem is usually solved at the velocity level [142]. Indeed, by differentiating (A.10) with respect to the time, it is possible to write

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{A.11}$$

where $\dot{\boldsymbol{a}}$ indicates the time derivative of $\boldsymbol{a}$, and the manipulator Jacobian $\boldsymbol{J} \triangleq \partial \boldsymbol{f}/\partial \boldsymbol{q} \in \mathbb{R}^{n\times m}$. If the Jacobian matrix is square and has full rank, the inverse kinematics problem is solved by means of the inverse of $\boldsymbol{J}$, i.e. $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\dot{\boldsymbol{x}}$. For redundant manipulators ($n < m$), the simplest strategy to solve the inverse kinematics problem is to minimize the norm of the joint velocity $\|\dot{\boldsymbol{q}}\|$. The inverse solution is then computed as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}}, \tag{A.12}$$

where $\boldsymbol{J}^{\dagger} \triangleq \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}})^{-1}$ is the Moore–Penrose pseudoinverse of $\boldsymbol{J}$.

Redundant manipulators are also able to execute the so-called null-space motions, i.e. movements of the robot body which do not affect the pose of the end-effector. Null-space motions are obtained by projecting extra tasks in the null-space of the Jacobian matrix. For example, given the end-effector task $\dot{\boldsymbol{x}}_{ee}$ and the null-space task $\dot{\boldsymbol{q}}_{ns}$, the inverse kinematic problem can be solved as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{x}}_{ee} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}_{ns}, \tag{A.13}$$

where the matrix $\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})$ projects $\dot{\boldsymbol{q}}_{ns}$ in the null-space of $\boldsymbol{J}$, and it is called a null-space projector.

A.2.3. Impedance control

The dynamics of a robotic manipulator in the absence of external forces (free-motion) can be written as [142]

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{A.14}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are respectively the joint positions, velocities, and accelerations, $\boldsymbol{B}(\boldsymbol{q})$ is the inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ accounts for the Coriolis and centrifugal effects, $\boldsymbol{F}_v$ represents the viscous friction, $\boldsymbol{g}(\boldsymbol{q})$ contains the gravitational terms, and $\boldsymbol{\tau}$ is the control torque. By choosing the control torque as

$$\boldsymbol{\tau} = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}), \tag{A.15}$$

the manipulator's dynamics in (A.14) becomes the linear and decoupled dynamical system

$$\ddot{\boldsymbol{q}} = \boldsymbol{u}. \tag{A.16}$$

The control law (A.15) is the so-called inverse dynamics control [142].

The inverse dynamics control requires to specify the control input $\boldsymbol{u}$. By choosing

$$\boldsymbol{u} = \ddot{\boldsymbol{q}}_d + \boldsymbol{K}(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{D}(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}), \tag{A.17}$$

the robot behaves as a spring–damper system that follows the desired trajectory specified by $\boldsymbol{q}_d$, $\dot{\boldsymbol{q}}_d$, and $\ddot{\boldsymbol{q}}_d$. The positive definite matrices $\boldsymbol{K}$ and $\boldsymbol{D}$ regulate the dynamic behavior of the robot. This choice of $\boldsymbol{u}$ is called impedance control, since equation (A.17) defines a mechanical impedance. Often in this work physical guidance is adopted to demonstrate a certain task to the robot. A simple control law that allows physical guidance is obtained by choosing $\boldsymbol{u} = \boldsymbol{0}$. In this way, the control input compensates for gravitational, friction, and Coriolis terms and lets the robot following external forces (human guidance). This choice of $\boldsymbol{u}$ is usually named gravity compensation or zero gravity control. Note that, in practice, the gravity compensation control transforms the robot into a weightless (or very light) object that is very easy to physically guide.

It can happen that the desired trajectory is specified in task space. In this case, inverse kinematics approaches can be used to obtain the desired trajectory in joint space. Alternatively, it is possible to define an impedance relationship $\boldsymbol{u}_c$ similar to (A.17) but in Cartesian space, and to transform $\boldsymbol{u}_c$ into a joint space torque using the relationship $\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{u}_c$ [142]. This solution is applied to control the KUKA light weight robot (LWR) [15].

A.3. Motion representation and regression

The goal of a regression technique is to estimate (regress) an output given a certain input and a vector of parameters, where the parameters vector is usually learned from a set of training data [16]. To this end, it is required to construct a continuous mapping between input and output spaces. When applied to robotics, regression techniques usually map the current position of the robot $\boldsymbol{x}_t$ into the desired position $\boldsymbol{x}_{t+1}$ or velocity $\dot{\boldsymbol{x}}_t$. The rest of the section gives an overview of three popular regression techniques—namely locally weighted projection regression, Gaussian mixture models, and hidden Markov models—widely used in this thesis work to represent human demonstrations.

A.3.1. Locally weighted projection regression

Locally weighted projection regression (LWPR) [154] is a non-linear regression approach that uses K linear models to predict the output $\boldsymbol{y}_i$ associated to a given input $\boldsymbol{x}_i$. Each of the K linear models represents a local region called Receptive Field (RF). Given an input point $\boldsymbol{x}_i$, the approach first computes the output of each RF using the linear relationship

$$\boldsymbol{y}_k = \boldsymbol{A}_k\boldsymbol{x}_i + \boldsymbol{b}_k,\ \forall k = 1,\dots,K. \tag{A.18}$$

The output $\boldsymbol{y}_i$ is then computed as

$$\boldsymbol{y}_i = \frac{\sum_{k=1}^{K} w_k(\boldsymbol{x}_i)\boldsymbol{y}_k}{\sum_{k=1}^{K} w_k(\boldsymbol{x}_i)}, \tag{A.19}$$

where the weight w_k determines the contribution of each RF. The weight w_k is usually the Gaussian kernel

$$w_k(\boldsymbol{x}_i) = \exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^{\mathrm{T}}\boldsymbol{W}_k(\boldsymbol{x}_i - \boldsymbol{\mu}_k)\right), \tag{A.20}$$

where $\boldsymbol{\mu}_k$ is the center of each RF and $\boldsymbol{W}_k$ is a positive semi-definite matrix. Once centers $\boldsymbol{\mu}_k$ are given, LWPR computes all the other parameters from the training data using an incremental least square algorithm.

A.3.2. Gaussian mixture models

Gaussian mixture models (GMM) [39] represent the joint probability $p(\boldsymbol{x};\boldsymbol{y})$ between input $\boldsymbol{x}$ and output $\boldsymbol{y}$ variables as a superposition of K Gaussian densities (components)

$$p(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(\omega_k)p(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} \pi_k\mathcal{N}(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k), \tag{A.21}$$

where each Gaussian component $\mathcal{N}(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$ has prior probability $p(\omega_k) = \pi_k$, mean $\boldsymbol{\mu}_k$, and covariance matrix $\boldsymbol{\Sigma}_k$. Note that the priors π_k satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K}\pi_k = 1$, while the means and covariance matrices in (A.21) are defined as

$$\boldsymbol{\mu}_k = \begin{bmatrix} {}^{x}\boldsymbol{\mu}_k \\ {}^{y}\boldsymbol{\mu}_k \end{bmatrix}, \quad \boldsymbol{\Sigma}_k = \begin{bmatrix} {}^{x}\boldsymbol{\Sigma}_k & {}^{xy}\boldsymbol{\Sigma}_k \\ {}^{yx}\boldsymbol{\Sigma}_k & {}^{y}\boldsymbol{\Sigma}_k \end{bmatrix}. \tag{A.22}$$

GMM parameters $\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ are learned from training data using the expectation–maximization algorithm [16].

Gaussian mixture regression (GMR) [39] is a non-linear regression technique that exploits the joint probability $p(\boldsymbol{x};\boldsymbol{y}|\boldsymbol{\theta})$ in (A.21) to compute the conditional distribution of the output $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$. In particular, given a query point $\boldsymbol{x}_i$, the conditional mean $\boldsymbol{y}_i$ and covariance matrix $\boldsymbol{\Sigma}_{y_i}$ are

$$\begin{aligned}\boldsymbol{y}_i &= \sum_{k=1}^{K} h_k(\boldsymbol{x}_i)\left({}^{yx}\boldsymbol{\Sigma}_k({}^{x}\boldsymbol{\Sigma}_k)^{-1}(\boldsymbol{x}_i - {}^{x}\boldsymbol{\mu}_k) + {}^{y}\boldsymbol{\mu}_k\right)\\ \boldsymbol{\Sigma}_{y_i} &= \sum_{k=1}^{K} h_k^2(\boldsymbol{x}_i)\left({}^{y}\boldsymbol{\Sigma}_k - {}^{yx}\boldsymbol{\Sigma}_k({}^{x}\boldsymbol{\Sigma}_k)^{-1}\,{}^{xy}\boldsymbol{\Sigma}_k\right)\end{aligned} \tag{A.23}$$

where

$$h_k(\boldsymbol{x}_i) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i|{}^{x}\boldsymbol{\mu}_k, {}^{x}\boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j \mathcal{N}(\boldsymbol{x}_i|{}^{x}\boldsymbol{\mu}_j, {}^{x}\boldsymbol{\Sigma}_j)}. \tag{A.24}$$

A.3.3. Hidden Markov models

Hidden Markov Models (HMM) [123] are a stochastic approach widely used to encode robot's skills from demonstrations (motion primitives), to retrieve the desired trajectory, and to recognize motion primitives from incoming data [69, 90, 135]. An HMM is described by the set of L hidden states $\boldsymbol{S} = \{s_1,\dots,s_L\}$, the set of M observable output symbols $\boldsymbol{O} = \{\boldsymbol{o}_1,\dots,\boldsymbol{o}_M\}$, the initial state probability $\boldsymbol{\pi} = \{\pi_1,\dots,\pi_L\}$, the $L\times L$ state transition probability matrix $\boldsymbol{A} = \{a_{ij}\}$, and the observation symbols probability distribution $\boldsymbol{B} = \{\boldsymbol{b}_1(\boldsymbol{o}),\dots,\boldsymbol{b}_L(\boldsymbol{o})\}$. When the observations $\boldsymbol{o}$ are continuous, a common choice is to represent the observation probability distribution as a mixture of M Gaussians

$$\boldsymbol{b}_j = \sum_{k=1}^{M} c_{jk}\mathcal{N}(\boldsymbol{o}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}), \tag{A.25}$$

where c_{jk} is the mixture coefficient (prior probability), $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\Sigma}_{jk}$ are respectively the mean and covariance matrix of the k-th component in the state s_j. The set of model parameters is usually indicated with $\boldsymbol{\lambda} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. Given a set of observations ${}^{s}\boldsymbol{O}$, HMM parameters $\boldsymbol{\lambda}$ are learned using the Baum-Welch algorithm [123].

The discrete nature of states in HMM results in generation of stepwise motion sequences. To overcome this problem, the work in [91] proposes a modified learning and generation approach. In particular, the learning algorithm works by introducing a normalized time variable for each state. This variable linearly varies from 0 (entering in the state) to 1 (leaving the state). Then, the correlation between temporal and spatial data is learned as follows. HMM parameters $\boldsymbol{\lambda}$ are firstly learned from a time series of spatial data (motion sequence) ${}^{s}\boldsymbol{O} = \{{}^{s}\boldsymbol{o}(t)\}$, using the Baum-Welch algorithm [123]. Secondly, the optimal state sequence $\boldsymbol{Q}^* = \{q(t)\}$ for the motion sequence ${}^{s}\boldsymbol{O}$ is calculated with the Viterbi algorithm [123]. From the optimal sequence $\boldsymbol{Q}^*$, the relative temporal sequence ${}^{t}\boldsymbol{O} = \{{}^{t}o(t)\}$, having mean equal to 0.5 for each state, is calculated. Finally, from ${}^{s}\boldsymbol{O}$ and ${}^{t}\boldsymbol{O}$, the covariance ${}^{ts}\boldsymbol{\Sigma}$ for each state is calculated.

The generation algorithm consists of four steps. Firstly, a state sequence is generated deterministically. The initial state $q(1)$ is chosen considering the initial state probability

$$q(1) = arg\max_i \boldsymbol{\pi}. \tag{A.26}$$

Then, for each state i, the expected duration to stay at i is calculated as $d = 1/(1-a_{ij})$, where a_{ij} is the probability to transit from state i to $j \neq i$. After staying in state i for d steps, the next state is chosen as

$$q(t+1) = arg\max_j a_{q(t)j}. \tag{A.27}$$

Secondly, from the state sequence $\boldsymbol{Q}$, the relative temporal sequence ${}^{t}\boldsymbol{O}$ is calculated. For this temporal sequence, the *responsibility* $\gamma_i(t)$ for each state i is calculated as

$$\gamma_i(t) = \frac{\alpha_i({}^{t}o(t))\beta_i({}^{t}o(t))}{\sum_{l=1}^{L} \alpha_l({}^{t}o(t))\beta_l({}^{t}o(t))}, \quad \forall l = 1, \dots, L, \tag{A.28}$$

where $\alpha_i({}^{t}o(t))$ and $\beta_i({}^{t}o(t))$ are the HMM forward and backward variables [123] for the temporal sequence. The responsibility $\gamma_i(t)$ represents the probability of being in state i at time t for the observation sequence ${}^{t}\boldsymbol{O}$.

Then, from $\boldsymbol{Q}$ and ${}^{t}\boldsymbol{O}$, a sequence of spatial data is calculated. For each time step t, a mixture of K Gaussians in the state $i = q(t)$ is considered, where the mean and covariance matrix of the generic k-th Gaussian at state i are

$$\boldsymbol{\mu}_{ik} = \begin{bmatrix} {}^{t}\mu_{ik} \\ {}^{s}\boldsymbol{\mu}_{ik} \end{bmatrix}, \quad \boldsymbol{\Sigma}_{ik} = \begin{bmatrix} {}^{tt}\boldsymbol{\Sigma}_{ik} & {}^{ts}\boldsymbol{\Sigma}_{ik} \\ {}^{st}\boldsymbol{\Sigma}_{ik} & {}^{ss}\boldsymbol{\Sigma}_{ik} \end{bmatrix}.$$

The conditional spatial data for each state i is calculated using GMR [39]

$${}^{s}\boldsymbol{o}_i(t) = \sum_{k=1}^{K} c_{ik} \left\{ {}^{s}\boldsymbol{\mu}_{ik} + \frac{{}^{ts}\boldsymbol{\Sigma}_{ik}}{{}^{tt}\boldsymbol{\Sigma}_{ik}} ({}^{t}\boldsymbol{o}(t) - {}^{t}\mu_{ik}) \right\}, \tag{A.29}$$

where i is used instead of $q(t)$ to simplify the notation. Finally, the conditional expectation ${}^{s}\boldsymbol{o}(t)$, representing the generated smooth trajectory, is generated considering the responsibility of each state as

$${}^{s}\boldsymbol{o}(t) = \sum_{l=1}^{L} \gamma_i(t)\, {}^{s}\boldsymbol{o}_i(t). \tag{A.30}$$

Incremental motion refinement

The goal of incremental learning of motion primitives is to update the previous knowledge of motion primitives as new demonstrations are provided, without keeping all the training data in the dataset. A forgetting factor is often used to avoid that the learning algorithm becomes insensitive to new data when the data set becomes large. An approach for incremental learning of HMM is proposed in [91] and summarized as follows.

The main idea is to use just two demonstrations (${}^{s}\boldsymbol{O}^{d}$, $d = 1,2$): one is the new demonstration provided by the user, the other is the smooth trajectory generated from the current HMM model using the previously described approach. Given the forgetting factor $0 < \eta < 1$, the weighting term for the new demonstration is chosen as $w^1 = \eta$, while the weighting term for the generated motion trajectory is $w^2 = 1 - \eta$. The new HMM parameters $\hat{\boldsymbol{\lambda}}$ are updated using the old ones $\boldsymbol{\lambda}$ and the demonstrations as follows

$$\begin{aligned}
\hat{\pi}_i &= \textstyle\sum_{d=1}^{2} w^d \gamma_i^d(1), \\
\hat{a}_{ij} &= \frac{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d-1} \xi_{ij}^d(t)}{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d-1} \gamma_i^d(t)}, \\
\hat{c}_{ik} &= \frac{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_{ik}^d(t)}{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_i^d(t)}, \\
{}^{s}\hat{\boldsymbol{\mu}}_{ik} &= \frac{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_{ik}^d(t)\, {}^{s}\boldsymbol{o}^d(t)}{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_{ik}^d(t)}, \\
{}^{ss}\hat{\boldsymbol{\Sigma}}_{ik} &= \frac{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_{ik}^d(t)({}^{s}\boldsymbol{o}^d(t) - {}^{s}\boldsymbol{\mu}_{ik})({}^{s}\boldsymbol{o}^d(t) - {}^{s}\boldsymbol{\mu}_{ik})^T}{\sum_{d=1}^{2} w^d \sum_{t=1}^{T^d} \gamma_{ik}^d(t)}.
\end{aligned}$$

In the former equations, γ_i^d is the probability of being in state i at time t for the observation sequence ${}^{s}\boldsymbol{O}^{d}$, T^d is the time duration of the observation sequence ${}^{s}\boldsymbol{O}^{d}$, ξ_{ij} is the probability of being in state i at time t and in state j at time $t+1$ for the observation sequence ${}^{s}\boldsymbol{O}^{d}$, and $\gamma_{ik}^d(t)$ is the probability of being in state i at time t with the k-th mixture component accounting for the observation ${}^{s}\boldsymbol{o}^{d}$.

APPENDIX B

Comparison of Invariant Motion Representations

This appendix describes theoretical relationships and differences between the DHB representation presented in Section 3.2.2 and two state-of-the-art bidirectional invariant representations, namely EFS [155] and DS[1] [42]. EFS and DS representations transform velocities and their time derivatives into invariants, as shown in Table B.1, where the time dependencies are omitted to simplify the notation. They are compared with velocity-based DHB invariants, since position and velocity-based DHB invariants are practically the same and the same properties hold for both the representations, as stated in Section 3.2.2. The results of the analysis presented in this section are summarized in Table B.2.

Table B.1.: EFS and DS representations

EFS invariants [155]
$e_\omega^1 = \pm\|\boldsymbol{\omega}\|,\ e_\omega^2 = \pm\frac{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|}{\|\boldsymbol{\omega}\|^2},\ e_\omega^3 = \pm\frac{\|(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})\times(\boldsymbol{\omega}\times\ddot{\boldsymbol{\omega}})\|}{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}$
$e_v^1 = \pm\|\mathbf{v}\|,\quad e_v^2 = \pm\frac{\|\mathbf{v}\times\dot{\mathbf{v}}\|}{\|\mathbf{v}\|^2},\quad e_v^3 = \pm\frac{\|(\mathbf{v}\times\dot{\mathbf{v}})\times(\mathbf{v}\times\ddot{\mathbf{v}})\|}{\|\mathbf{v}\times\dot{\mathbf{v}}\|^2}$
DS invariants [42]
$d_\omega^1 = \pm\|\boldsymbol{\omega}\|,\ d_\omega^2 = \pm\frac{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|}{\|\boldsymbol{\omega}\|^2},\ d_\omega^3 = \pm\frac{\|\boldsymbol{\omega}\|}{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}\lvert(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})\cdot\ddot{\boldsymbol{\omega}}\rvert$
$d_v^1 = \pm\frac{\mathbf{v}\cdot\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|},\quad d_v^2 = \pm\frac{\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}}{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|}\cdot\frac{\|\boldsymbol{\omega}\|^2(\dot{\boldsymbol{\omega}}\times\mathbf{v}+\boldsymbol{\omega}\times\dot{\mathbf{v}})-2(\boldsymbol{\omega}\times\mathbf{v})\cdot(\boldsymbol{\omega}\cdot\dot{\boldsymbol{\omega}})}{\|\boldsymbol{\omega}\|^4}$
$d_v^3 = \mp\frac{[\dot{\boldsymbol{\omega}}\times(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})+\boldsymbol{\omega}\times(\boldsymbol{\omega}\times\ddot{\boldsymbol{\omega}})]\cdot\left[\|\boldsymbol{\omega}\|^2(\dot{\boldsymbol{\omega}}\times\mathbf{v}+\boldsymbol{\omega}\times\dot{\mathbf{v}})-2\boldsymbol{\omega}\cdot\dot{\boldsymbol{\omega}}\cdot(\boldsymbol{\omega}\times\mathbf{v})\right]}{\|\boldsymbol{\omega}\|^3\cdot\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}$
$\mp\frac{[\boldsymbol{\omega}\times(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})]\cdot\left[\|\boldsymbol{\omega}\|^2(\ddot{\boldsymbol{\omega}}\times\mathbf{v}+2\dot{\boldsymbol{\omega}}\times\dot{\mathbf{v}}+\boldsymbol{\omega}\times\ddot{\mathbf{v}})-2(\|\dot{\boldsymbol{\omega}}\|^2+\boldsymbol{\omega}\cdot\dot{\boldsymbol{\omega}})(\boldsymbol{\omega}\times\mathbf{v})\right]}{\|\boldsymbol{\omega}\|^3\cdot\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}$
$\pm\left[\frac{3(\boldsymbol{\omega}\cdot\dot{\boldsymbol{\omega}})}{2\|\boldsymbol{\omega}\|^2}+\frac{(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})\cdot(\boldsymbol{\omega}\times\ddot{\boldsymbol{\omega}})}{\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}\right]\cdot\frac{[\boldsymbol{\omega}\times(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})]\cdot\left[\|\boldsymbol{\omega}\|^2(\dot{\boldsymbol{\omega}}\times\mathbf{v}-\boldsymbol{\omega}\times\dot{\mathbf{v}})-2(\boldsymbol{\omega}\cdot\dot{\boldsymbol{\omega}}(\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}})\right]}{\|\boldsymbol{\omega}\|^3\cdot\|\boldsymbol{\omega}\times\dot{\boldsymbol{\omega}}\|^2}$

[1]For simplicity, the acronym of the author name (DS) is used to refer the representation in [42].

Table B.2.: Overview and properties of DS, EFS, and DHB representations.

			Numerically	Invariant to			
Representation	Physical meaning	Bidirectional	robust	Initial pose	reference point	viewpoint	time/linear/angular scale
DS	Motion along ISA and motion of ISA	✓	–	✓	✓	✓	✓/✓/✓
EFS	Velocities (magnitude), orientation of FS frames	✓	–	✓	–	✓	✓/✓/✓
DHB	Discrete time approximation of EFS	✓	✓	✓	–	✓	✓/✓/✓

B.1. Theoretical comparison

B.1.1. DHB and DS representations

DS [42] is a bidirectional invariant representation of rigid body motions, constructed by means of the Instantaneous Screw Axis (ISA) [101]. Two invariants represent the translational velocity along the ISA and the rotational velocity about the ISA. Four other invariant values describe the motion of the ISA between two consecutive time instants. Given the twist, i.e. linear $\mathbf{v}_t$ and angular $\boldsymbol{\omega}_t$ velocities, in each time instant, the six invariants are computed as shown in Table B.1, where the sign of each invariant is chosen to avoid discontinuities between consecutive time instants. In the continuous time domain, i.e. for the sampling time $\Delta t \longrightarrow 0$, the relationships between DHB and DS are $m_\omega = d^1_\omega$, $\theta^1_\omega \approx d^2_\omega \Delta t$, and $\theta^2_\omega \approx d^3_\omega \Delta t$. These relationships are proven as follows.

The relationship $m_\omega = d^1_\omega$ derives from (3.37) and d^1_ω in Table B.1. In order to show that $\theta^1_\omega \approx d^2_\omega \Delta t$, one has to notice that, for $\Delta t \longrightarrow 0$, it is possible to neglect the arc tangent in (3.45). Hence, θ^1_ω in (3.40) can be re-written as

$$\theta^1_\omega \approx \frac{\|\boldsymbol{\omega}_t \times \boldsymbol{\omega}_{t+1}\|}{\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_{t+1}} = \frac{\|\boldsymbol{\omega}_t \times (\boldsymbol{\omega}_t + \Delta\boldsymbol{\omega}_t)\|}{\boldsymbol{\omega}_t \cdot (\boldsymbol{\omega}_t + \Delta\boldsymbol{\omega}_t)} \approx \frac{\|\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t\|}{\|\boldsymbol{\omega}_t\|^2} \frac{\Delta t}{\Delta t} \approx \frac{\|\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t\|}{\|\boldsymbol{\omega}_t\|^2} \Delta t = d^2_\omega \Delta t. \tag{B.1}$$

In order to show that $\theta^2_\omega \approx d^3_\omega \Delta t$, recall that $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$ and that $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b})$. θ^2_ω in (3.40) can be re-written as

$$\theta^2_\omega = \arctan\left(\frac{\|\boldsymbol{\omega}_{t+1}\| \boldsymbol{\omega}_{t+2} \cdot (\boldsymbol{\omega}_{t+1} \times \boldsymbol{\omega}_t)}{(\boldsymbol{\omega}_{t+1} \times \boldsymbol{\omega}_t) \cdot (\boldsymbol{\omega}_{t+1} \times \boldsymbol{\omega}_{t+2})}\right) = \arctan\left(\frac{\|\boldsymbol{\omega}_{t+1}\| (\boldsymbol{\omega}_t \times \boldsymbol{\omega}_{t+1}) \cdot \boldsymbol{\omega}_{t+2}}{(\boldsymbol{\omega}_t \times \boldsymbol{\omega}_{t+1}) \cdot (\boldsymbol{\omega}_{t+1} \times \boldsymbol{\omega}_{t+2})}\right). \tag{B.2}$$

The denominator of (B.2) can be re-written as

$$\begin{aligned} &(\boldsymbol{\omega}_t \times \boldsymbol{\omega}_{t+1}) \cdot (\boldsymbol{\omega}_{t+1} \times \boldsymbol{\omega}_{t+2}) \approx (\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t) \cdot [(\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t) \times (\boldsymbol{\omega}_t \times 2\Delta\boldsymbol{\omega}_t)] \\ &= (\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t) \cdot [2(\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t) - (\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t)] \frac{\Delta t^2}{\Delta t^2} \approx (\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t) \cdot (\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t) \Delta t^2 = \|\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t\|^2 \Delta t^2. \end{aligned} \tag{B.3}$$

Considering that $\ddot{\boldsymbol{a}}_t \approx (\boldsymbol{a}_{t+2} + \boldsymbol{a}_t)/\Delta t^2$, the numerator of (B.2) can be re-written as

$$\|\boldsymbol{\omega}_{t+1}\| (\boldsymbol{\omega}_t \times \boldsymbol{\omega}_{t+1}) \cdot \boldsymbol{\omega}_{t+2} \approx \|\boldsymbol{\omega}_t\| (\boldsymbol{\omega}_t \times \Delta\boldsymbol{\omega}_t) \cdot (\ddot{\boldsymbol{\omega}}_t \Delta t^2 - \boldsymbol{\omega}_t) \approx \|\boldsymbol{\omega}_t\| (\boldsymbol{\omega}_t \times \dot{\boldsymbol{\omega}}_t) \cdot \ddot{\boldsymbol{\omega}}_t \Delta t^3. \tag{B.4}$$

Finally, combining (B.3), (B.4), and (B.2), and neglecting the arc tangent, it results that $\theta^2_\omega \approx d^3_\omega \Delta t$ for $\Delta t \longrightarrow 0$.

Described relationships allow to conclude that the DHB invariant values related to the angular velocity represent the angular motion of the ISA in the discrete time domain. DHB invariants describe the motion of the ISA using angular velocities sampled at consecutive time instants, while DS invariants use high-order time derivatives. DS invariant values related to the linear velocity are obtained by projecting the linear velocity along the ISA, which guarantees the invariance to the reference point used to describe the translation [42]. As discussed in Section 3.2.2, the invariance to the reference point is not guaranteed by DHB and EFS representations.

B.1.2. DHB and EFS representations

The original Frenet–Serret (FS) representation [83] consists of three invariant values, corresponding to e_v^1, e_v^2, and e_v^3 in Table B.1. e_v^1 represents the linear velocity along the tangent axis of the Frenet–Serret frame, e_v^2 and e_v^3 describe the change in the orientation of the FS frame. e_v^2 and e_v^3 are closely related to the curvature κ and torsion τ of a space curve, i.e. $e_v^2 = \pm\kappa\|\mathbf{v}\|$ and $e_v^3 = \pm\tau\|\mathbf{v}\|$. Recall that the curvature describes the change of the orientation angle of the tangent of a space curve per unit arc length, while the torsion describes the change of the orientation of the tangent plane of a space curve per unit arc length [83].

EFS [155] extends the FS representation by considering the orientation of the rigid body. In EFS a second FS frame is attached to the rigid body and three more invariants—e_ω^1, e_ω^2, and e_ω^3 in Table B.1—are used to describe the rotation of the rigid body. The original trajectory can be reconstructed from EFS invariants by applying the method in [164] to both the FS frames. From Table B.1, it is straightforward to show that $d_\omega^1 = e_\omega^1$, $d_\omega^2 = e_\omega^2$, and $d_\omega^3 = e_\omega^3$. Another interesting property is that EFS and DS are exactly the same representation in case of pure translations. In this case, in fact, $\boldsymbol{\omega} = \mathbf{0}$ and only three invariants are defined. According to [42] and Table B.1, it holds that $d_v^1 = e_v^1$, $d_\omega^2 = e_v^2$, and $d_\omega^3 = e_\omega^3$.

In the continuous time domain, DHB and EFS are almost the same apart from a scaling factor. Indeed, it holds that $m_u = e_u^1$, $\theta_u^1 \approx e_u^2\Delta t$, and $\theta_u^2 \approx e_u^3\Delta t$ for $u = v, \omega$. The relationship $m_v = e_v^1$ derives from (3.36) and e_v^1 in Table B.1. The relationship $m_\omega = e_\omega^1$ derives from (3.37) and e_ω^1 in Table B.1, while $\theta_\omega^1 \approx e_\omega^2\Delta t$ derives from (B.1) recalling that $e_\omega^2 = d_\omega^2$. $\theta_v^1 \approx e_v^2\Delta t$ can be proven by following similar steps as in (B.1) and considering e_v^2 in Table B.1. The relationship $\theta_v^2 \approx e_v^3\Delta t$ and $\theta_\omega^2 \approx e_\omega^3\Delta t$ are obtained by following similar steps as in (B.3) and (B.4), and recalling that $(\mathbf{a}\times\mathbf{b})\times(\mathbf{a}\times\mathbf{c}) = [\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})]\,\mathbf{a}$.

Given these relationships, it is possible to conclude that DHB is a discrete time version of EFS. EFS invariants describe the motion of the two FS frames assuming a continuous time domain and the effects of the numerical implementation are simply neglected. In contrast, DHB invariants describe the motion of the two (FS) frames directly assuming a discrete time domain. DHB uses velocities sampled at consecutive time instants instead of high-order time derivatives. DHB and EFS share the same invariance properties (see Table B.2), because both describe the motion of the two FS frames. In addition, DHB representation is also numerically robust because there is a one-to-one mapping between the representation and its numerical implementation.

Note that a robust discrete time approximation of FS invariants is proposed in [163]. Compared to [163], DHB representation considers also the orientation of the rigid body, uses less consecutive samples (3 instead of 5), and provides an accurate approach to reconstruct the motion. Finally, it is straightforward to show that θ_v^1, θ_v^2, θ_ω^1, and θ_ω^2 depend on the sampling time Δt. The sampling time dependency does not cause problems in usual cases, since most of the sensors provides measurements with relatively constant sampling time. In case the trajectories are sampled at very different rates, one can simply divide these invariants by the sampling time Δt. Dividing by Δt corresponds to a numerical differentiation step, and it increases the noise sensitivity of DHB invariants. As a general rule, it is preferable to remove the dependency on the sampling time only if it is needed, i.e. if trajectories are sampled at different rates.

B.2. Experimental comparison

B.2.1. Trajectory reconstruction error

DHB, DS, and EFS are bidirectional invariant representations, i.e. Cartesian twists can be retrieved from their representations. Original velocities are reconstructed from a DHB descriptor

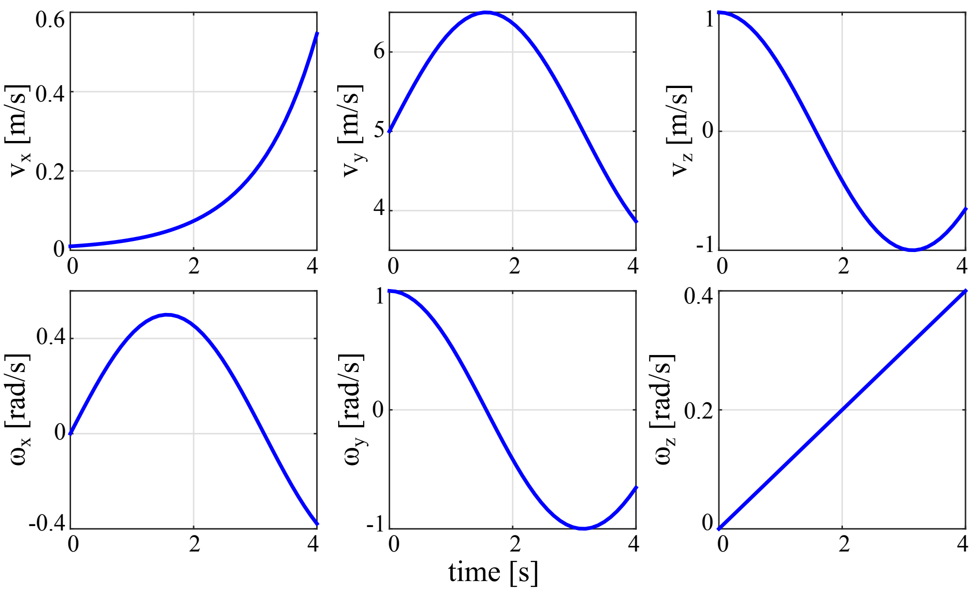


Figure B.1.: Synthetic twist trajectory: $v_x = 0.01\exp(t)$, $v_y = 5 + 1.5\sin(t)$, $v_z = \cos(t)$, $\omega_x = 0.5\sin(t)$, $\omega_y = \cos(t)$ and $\omega_z = 0.1t$, $t = 0,\dots,4\,\mathrm{s}$.

by applying the algorithm in Section 3.2.2. The reconstruction performance of all the representations is tested by using the synthetic twist trajectory in Figure B.1. The reconstruction error is computed as the root mean square error (RMSE) between the original twists in Figure B.1 and the twists reconstructed from each invariant representation. The results are shown in Figure B.2 using three different sampling times ($\Delta t = 0.1$, 0.01, and 0.001 s). One can see a non-negligible error with DS and EFS descriptors. The accuracy of DS and EFS depends on the sampling time, i.e. the smaller the sampling time the smaller the error. DS and EFS representations, in fact, use high-order derivatives which are affected by round-off errors. Moreover, numerical integration steps are required to reconstruct Cartesian twists from both DS and EFS representations. The linear RMSE for DS representation is always bigger than the linear RMSE for EFS invariants. This is because d_v^2 and d_v^3 are an approximated representation of the kinematics of the ISA [42]. On the other hand, the DHB invariants offer a high reconstruction accuracy for each value of Δt. Figure B.2 shows a negligible increase of reconstruction error for smaller sampling times, which is due to the finite precision of the computing machine. In particular, a smaller sampling time generates more twist samples and more invariant values, and more products have to be computed in (3.56) to reconstruct the motion, which increases errors due to the finite precision.

B.2.2. Noise sensitivity

In order to show the noise sensitivity of invariant descriptors, a Gaussian noise with increasing power P_{noise} is added to the data in Figure B.1. The Gaussian noise is generated by considering a decreasing signal noise ratio (snr) from 100 dB to 50 dB, where $snr = P_{signal}/P_{noise}$. To measure the noise sensitivity of each representation, the invariant representation $\{i_n\}_t$ of the noisy twist trajectories is computed. The invariant representation $\{i\}_t$ of the noiseless trajectories is then point-wise subtracted to $\{i_n\}_t$. The resulting set of samples $\{r_n\}_t$ represent the residual noise. The residual signal to noise ratio, i.e. the ratio between the original signal power $P_{\{i\}}$ and the residual noise power $P_{\{r_n\}}$, is used to compactly represent the noise sensitivity. The described procedure is repeated 100 times for each value of the snr and for each invariant representation.

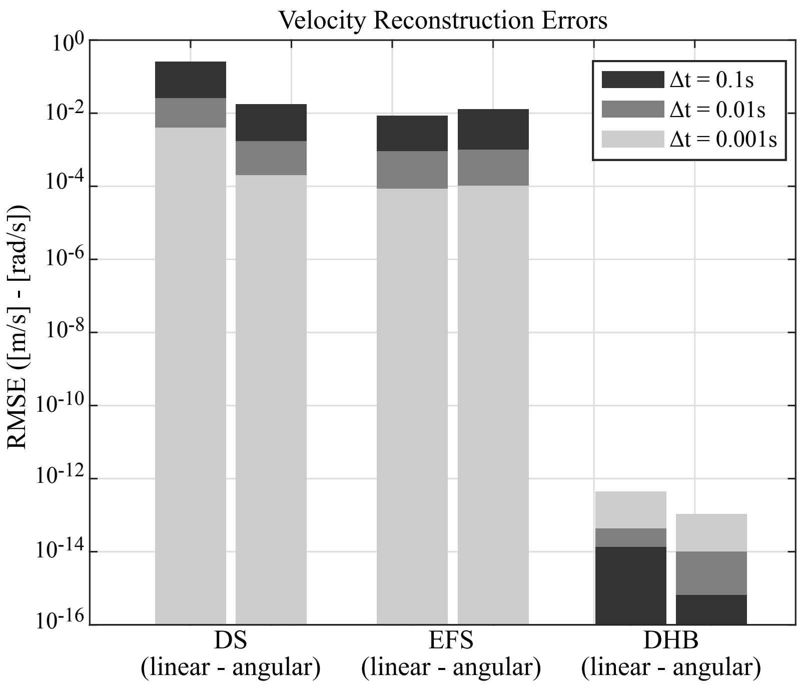


Figure B.2.: Errors between the twists in Figure B.1 and the twists reconstructed from DS, EFS, and DHB for different sampling times. Note the logarithmic scale on the ordinates.

Figure B.3 shows the mean and standard deviation of the residual snr. The DHB representation exhibits a reduced noise sensitivity, compared to DS and EFS representations. The reason is that DHB invariants lie at velocity level, while DS and EFS invariants lie at jerk level. The numerical computation of high-order derivatives is sensitive to the noise in the data and round-off errors [34]. DS invariants show the highest noise sensitivity. Indeed, the projection of the linear velocity (and its time derivatives) along the ISA axis increases the noise, especially in d_v^2 and d_v^3 in Table B.1.

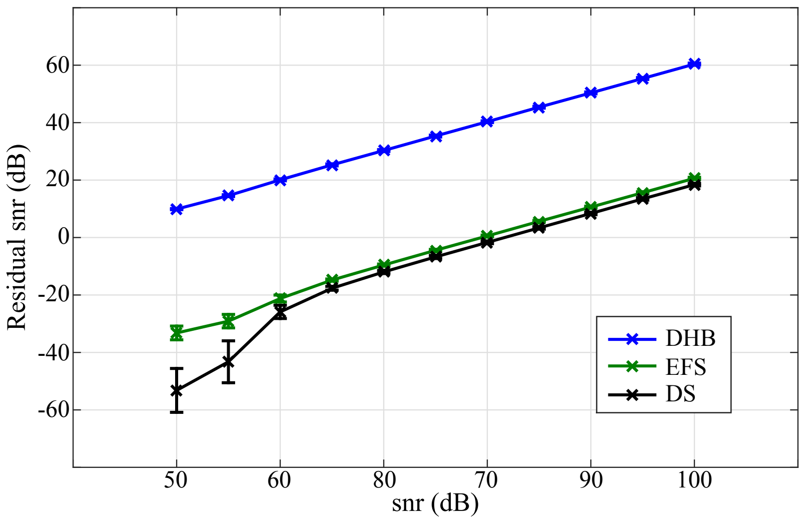


Figure B.3.: Noise sensitivity of DHB, DS, and EFS when a Gaussian noise with increasing power is applied to the twists in Figure B.1. Crosses represent the mean and bars represent the standard deviation of the residual snr over the 100 iterations.

Bibliography

[1] English letters dataset. www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-hmms/.

[2] LASA handwritten dataset. lasa.epfl.ch/sourcecode/.

[3] Microsoft research center action3d dataset. www.uow.edu.au/~wanqing/#MSRAction3DDatasets.

[4] NAO humanoid robot. www.ald.softbankrobotics.com/en/cool-robots/nao.

[5] Open Natural Interaction (OpenNI) library. www.openni.ru.

[6] Real-time URDF filter. https://github.com/blodow/realtime_urdf_filter.

[7] Virtual robot experimentation platform (V-Rep). www.coppeliarobotics.com/.

[8] Xsens MVN motion capture suit. www.xsens.com/products/xsens-mvn.

[9] S. An and D. Lee. Prioritized inverse kinematics using qr and cholesky decompositions. In *International Conference on Robotics and Automation*, pages 5062–5069, 2014.

[10] S. An and D. Lee. Prioritized inverse kinematics with multiple task definitions. In *International Conference on Robotics and Automation*, pages 1423–1430, 2015.

[11] A. S. Arefin, C. Riveros, R. Berretta, and P. Moscato. Gpu-fs-knn: A software tool for fast and scalable knn computation using gpus. *PLoS ONE*, 7(8):e44000, 2012.

[12] L. Bascetta, G. A. Magnani, P. Rocco, R. Migliorini, and M. Pelagatti. Anti-collision systems for robotic applications based on laser time-of-flight sensors. In *International conference on advanced intelligent mechatronics*, pages 278–284, 2010.

[13] A. Belardinelli, F. Pirri, and A. Carbone. Bottom-up gaze shifts and fixations learning by imitation. *Transactions on systems, man, and cybernetics, part B: cybernetics*, 37(2):256–271, 2007.

[14] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Springer handbook of robotics*, pages 1371–1394. 2008.

[15] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger. The kuka–dlr lightweight robot arm - a new reference platform for robotics research and manufacturing. In *International symposium on robotics*, pages 1–8, 2010.

[16] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[17] M. Black and D. Jepson. A probabilistic framework for matching temporal trajectories: Condensation-based recognition of gestures and expressions. In *European Conference on Computer Vision*, volume 1406 of *Lecture Notes in Computer Science*, pages 909–924. Springer, 1998.

[18] C. Blocher, M. Saveriano, and D. Lee. Learning stable dynamical systems using contraction theory. In *International Conference on Ubiquitous Robots and Ambient Intelligence*, pages 124–129, 2017.

[19] A. Borji, M. N. Ahmadabadi, B. N. Araabi, and M. Hamidi. Online learning of task-driven object-based visual attention control. *Image and vision computing*, 28(7):1130–1145, 2010.

[20] M. M. Botvinick, T. S. Braver, D. M. Barch, C. S. Carter, and J. D. Cohen. Conflict monitoring and cognitive control. *Psychological review*, 108(3):624–652, 2001.

[21] C. Breazeal and M. Berlin. Spatial scaffolding for sociable robot learning. In *AAAI Conference on artificial intelligence*, pages 1268–1273, 2008.

[22] D. Bristow, M. Tharayil, and A. Alleyne. A survey of iterative learning control. *Control Systems Magazine*, 25(3):96–114, 2006.

[23] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *The international journal of robotics research*, 21(12):1031–1052, 2002.

[24] X. Broquère, A. Finzi, J. Mainprice, S. Rossi, D. Sidobre, and M. Staffa. An attentional approach to human–robot interactive manipulation. *International Journal of Social Robotics*, 6(4):533–553, 2014.

[25] R. Caccavale, J. Cacace, M. Fiore, R. Alami, and A. Finzi. Attentional supervision of human-robot collaborative plans. In *International symposium on robot and human interactive communication*, pages 867–873, 2016.

[26] R. Caccavale and A. Finzi. Plan execution and attentional regulations for flexible human-robot interaction. In *International Conference on Systems, Man, and Cybernetics*, pages 2453–2458, 2015.

[27] R. Caccavale and A. Finzi. Flexible task execution and attentional regulations in human-robot interaction. *Transactions on cognitive and developmental systems*, 9(1):68–79, 2017.

[28] R. Caccavale, A. Finzi, D. Lee, and M. Saveriano. Integrated task learning and kinesthetic teaching for human-robot cooperation. In *Italian Workshop on Artificial Intelligence and Robotics*, 2016.

[29] R. Caccavale, E. Leone, L. Lucignano, S. Rossi, M. Staffa, and A. Finzi. Attentional regulations in a situated human-robot dialogue. In *International symposium on robot and human interactive communication*, pages 844–849, 2014.

[30] R. Caccavale, M. Saveriano, G. A. Fontanelli, F. Ficuciello, D. Lee, and A. Finzi. Imitation learning and attentional supervision of dual-arm structured tasks. In *International conference on development and learning and on epigenetic robotics*, 2017.

[31] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent service robotics*, 9(1):1–29, 2016.

[32] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *International conference on Human–robot interaction*, pages 255–262, 2007.

[33] S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humanoid robot. *Transactions on systems, man and cybernetics, part B: cybernetics*, 37(2):286–298, 2007.

[34] R. Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011:1–12, 2011.

[35] R. Chaudhry, F. Ofli, G. Kurillo, R. Bajcsy, and R. Vidal. Bio-inspired dynamic 3d discriminative skeletal features for human action recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 471–478, 2013.

[36] S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transaction on Robotics and Automation*, 13(3):398–410, 1997.

[37] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT press, 2005.

[38] M. Cirillo, L. Karlsson, and A. Saffiotti. Human-aware task planning: An application to mobile robots. *Transactions on Intelligent Systems and Technology*, 1(2), 2010.

[39] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4(1):129–145, 1996.

[40] R. P. Cooper and T. Shallice. Hierarchical schemas and goals in the control of sequential behavior. *Psychological review*, 113(4):887–916, 2006.

[41] G. De Maria, P. Falco, C. Natale, and S. Pirozzi. Integrated force/tactile sensing: The enabling technology for slipping detection and avoidance. In *International Conference on Robotics and Automation*, pages 3883–3889, 2015.

[42] J. De Schutter. Invariant description of rigid body motion trajectories. *Journal of Mechanisms and Robotics*, 2(1):1–9, 2010.

[43] J. De Schutter, E. Di Lello, J. F. M. De Schutter, R. Matthysen, T. Benoit, and T. De Laet. Recognition of 6 dof rigid body motion trajectories using a coordinate-free representation. In *International Conference on Robotics and Automation*, pages 2071–2078, 2011.

[44] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Transaction of the ASME journal of applied mechanics*, 22(2):215–221, 1965.

[45] S. Dieleman, J. De Fauw, and K. Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *International Conference on Machine Learning*, pages 1889–1898, 2016.

[46] R. Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and autonomous systems*, 47(2):109–116, 2004.

[47] P. M. Engel and M. R. Heinen. Incremental learning of multivariate gaussian mixture models. In *Advanvances in Artificial Intelligence*, pages 82–91. 2010.

[48] P. Falco and C. Natale. Low-level flexible planning for mobile manipulators: a distributed perception approach. *Advanced robotics*, 28(21):1431–1444, 2014.

[49] P. Falco, M. Saveriano, E. G. Hasany, N. H. Kirk, and D. Lee. A human action descriptor based on motion coordination. *Robotics and automation letters*, 2(2):811–818, 2017.

[50] F. Flacco and A. De Luca. Multiple depth/presence sensors: Integration and optimal placement for human/robot coexistence. In *International conference on robotics and automation*, pages 3916–3923, 2010.

[51] F. Flacco, T. Kroeger, A. De Luca, and O. Khatib. A depth space approach for evaluating distance to objects. *Journal of Intelligent and Robotic Systems*, 80(1):7–22, 2015.

[52] F. Flacco, T. Kroger, A. De Luca, and O. Khatib. A depth space approach to human-robot collision avoidance. In *International conference on Robotics and Automation*, pages 338–345, 2012.

[53] A. Fod, M. J. Matarić, and O. C. Jenkins. Automated derivation of primitives for movement classification. *Autonomous robots*, 12(1):39–54, 2002.

[54] A. Gams, B. Nemec, A. J. Ijspeert, and A. Ude. Coupling movement primitives: Interaction with the environment and bimanual tasks. *Transactions on Robotics*, 30(4):816–830, 2014.

[55] S. Gams, A. J. Ijspeert, S. Schaal, and J. Lenarčič. On-line learning and modulation of periodic movements with nonlinear dynamical systems. *Autonomous Robots*, 27(1):3–23, 2009.

[56] S. Garrido-Jurado, R. Muñoz Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[57] G. Grioli, S. Wolf, M. Garabini, M. Catalano, E. Burdet, D. Caldwell, R. Carloni, W. Friedl, M. Grebenstein, M. Laffranchi, D. Lefeber, S. Stramigioli, N. Tsagarakis, M. van Damme, B. Vanderborght, A. Albu-Schaeffer, and A. Bicchi. Variable stiffness actuators: The user's point of view. *The International Journal of Robotics Research*, 34(6):727–743, 2015.

[58] S. Haddadin, A. Albu-Schäffer, A. De Luca, and G. Hirzinger. Collision detection and reaction: A contribution to safe physical human-robot interaction. In *International conference on Intelligent robots and systems*, pages 3356–3363, 2008.

[59] S. Haddadin, S. Belder, and A. Albu-Schäffer. Dynamic motion planning for robots in partially unknown environments. In *IFAC world congress*, pages 6842–6850, 2011.

[60] S. Haddadin, S. Haddadin, A. Khoury, T. Rokahr, S. Parusel, R. Burgkart, A. Bicchi, and A. Albu-Schäffer. On making robots understand safety: Embedding injury knowledge into control. *The International Journal of Robotics Research*, 31(13):1578–1602, 2012.

[61] S. Haddadin, H. Urbanek, S. Parusel, D. Burschka, J. Roßmann, A. Albu-Schäffer, and G. Hirzinger. Real-time reactive motion generation based on variable attractor dynamics and shaped velocities. In *International conference on intelligent robots and systems*, pages 3109–3116, 2010.

[62] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *International Conference on Robotics and Automation*, pages 1534–1539, 2009.

[63] K. Hu and D. Lee. Bipedal locomotion primitive learning, control and prediction from human data. *International symposium on robot control*, 45(22):536–542, 2012.

[64] K. Hu, C. Ott, and D. Lee. Online human walking imitation in task and joint space based on quadratic programming. In *International conference on robotics and automation*, pages 3458–3464, 2014.

[65] A. Ijspeert, J. Nakanishi, P Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[66] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems*, pages 1547–1554, 2003.

[67] A. J. Ijspeert, J. Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation*, pages 1398–1403, 2002.

[68] K. Ikuta, H. Ishii, and M. Nokatao. Safety evaluation method of design and control for human-care robot. *International Journal of Robotics Research*, 22(5):281–297, 20103.

[69] T. Inamura, I. Toshima, H. Tanie, and Y. Nakamura. Embodied symbol emergence based on mimesis theory. *The international journal of robotic research*, 23(4):363–377, 2004.

[70] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *European Conference on Computer Vision*, pages 343–356, 1996.

[71] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[72] Rainer Jäkel, Sven R. Schmidt-Rohr, Steffen W. Rühl, Alexander Kasper, Zhixing Xue, and Rüdiger Dillmann. Learning of planning models for dexterous manipulation based on human demonstrations. *International Journal of Social Robotics*, 4(4):437–448, 2012.

[73] K. B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, and D. A. Anisi. Collision avoidance with potential fields based on parallel processing of 3d-point cloud data on the gpu. In *International conference on robotics and automation*, pages 3250–3257, 2014.

[74] K. Kawamura, S. M. Gordon, P. Ratanswasd, E. Erdemir, and J. F. Hall. Implementation of cognitive control for a humanoid robot. *International journal of humanoid robotics*, 5(4):547–586, 2007.

[75] S. M. Khansari-Zadeh and A. Billard. BM: an iterative algorithm to learn stable non-linear dynamical systems with Gaussian mixture models. *International conference on robotics and automation*, pages 2381–2388, 2010.

[76] S. M. Khansari-Zadeh and A. Billard. Learning stable non-linear dynamical systems with Gaussian Mixture Models. *Trans. on Rob.*, 27(5):943–957, 2011.

[77] S. M. Khansari-Zadeh and A. Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous robots*, 32(4):433–454, 2012.

[78] S. M. Khansari-Zadeh and A Billard. Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics And Autonomous Systems*, 62(6):752–765, 2014.

[79] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[80] K. Khoshelham and S. Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[81] H. S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. *Transactions on pattern analysis and machine intelligence*, 38(1):14–29, 2015.

[82] K. Kronander, S. M. Khansari-Zadeh, and A. Billard. Incremental motion learning with locally modulated dynamical systems. *Robotics and Autonomous Systems*, 2015.

[83] W. Kühnel. *Differential geometry: curves - surfaces - manifolds*. American Mathematical Society, 2006.

[84] D. Kulić and E. A. Croft. Real-time safety for human–robot interaction. *Robotics and Autonomous Systems*, 54(1):1–12, 2006.

[85] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura. Incremental learning of full body motion primitives and their sequencing through human motion observation. *The international journal of robotics research*, 31(3):330–345, 2012.

[86] T. Kulvicius, M. Biehl, M. J. Aein, M. Tamosiunaite, and F. Wörgötter. Interaction learning for dynamic movement primitives used in cooperative robotic tasks. *Robotics and Autonomous Systems*, 61(12):1450–1459, 2013.

[87] B. Lacevic, P. Rocco, and A. M. Zanchettin. Safety assessment and control of robotic manipulators using danger field. *Transactions on Robotics*, 29(5):1257–1270, 2013.

[88] P. A. Lasota and J. A. Shah. Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 57(1):21–33, 2015.

[89] Y. LeCun. *Learning Invariant Feature Hierarchies*, pages 496–505. Springer, 2012.

[90] D. Lee and Y. Nakamura. Mimesis model from partial observations for a humanoid robot. *The international journal of robotic research*, 29(1):60–80, 2010.

[91] D. Lee and C. Ott. Incremental kinesthetic teaching of motion primitives using the motion refinement tube. *Autonomous robots*, 31(2):115–131, 2011.

[92] D. Lee, R. Soloperto, and M. Saveriano. Bidirectional invariant representation of rigid body motions and its application to gesture recognition and reproduction. *Autonomous Robots*, 2017.

[93] A. Lemme, F. Reinhart, K. Neumann, and J. J. Steil. Neural learning of vector fields for encoding stable dynamical systems. *Neurocomputing*, 141:3–14, 2014.

[94] W. Li, Z. Zhang, and Z. Liu. Action recognition based on a bag of 3d points. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–14, 2010.

[95] W. Lohmiller and J. J. E. Slotine. On contraction analysis for nonlinear systems. *Automatica*, 34(6):683–696, 1998.

[96] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2003.

[97] V. Magnanimo, M. Saveriano, S. Rossi, and D. Lee. A bayesian approach for task recognition and future human activity prediction. In *International Symposium on Robot and Human Interactive Communication*, pages 726–731, 2014.

[98] E. Magrini, F. Flacco, and A. De Luca. Estimation of contact forces using a virtual force sensor. In *International Conference on intelligent robots and systems*, pages 2126–2133, 2014.

[99] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstration. *Robotics and autonomous systems*, 74, Part A(12):97–107, 2015.

[100] Z.-C. Marton, R. B. Rusu, and M. Beetz. On fast surface reconstruction methods for large and noisy datasets. In *International Conference on Robotics and Automation*, pages 3218–3223, 2009.

[101] R. M. Murray, S. S. Sastry, and Z. Li. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1st edition, 1994.

[102] Y. Nagai. From bottom-up visual attention to robot action learning. In *International conference on development and learning*, pages 1–6, 2009.

[103] Y. Nakamura, H. Hanafusa, and T. Yoshikawa. Task-priority based redundancy control of robot manipulators. *International Journal of Robotic Research*, 6(2), 1987.

[104] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: an HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[105] B. Nemec, T. Petrič, and A. Ude. Force adaptation with recursive regression iterative learning controller. In *International Conference on Intelligent Robots and Systems*, pages 2835–2841, 2015.

[106] K. Neumann and J. J. Steil. Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robotics and autonomous systems*, 70(8):1–15, 2015.

[107] M. N. Nicolescu and M. J. Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *International joint conference on autonomous agents and multiagent systems*, pages 241–248, 2003.

[108] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics Science and Systems*, 2013.

[109] A. Nordmann, C. Emmerich, S. Rüther, A. Lemme, S. Wrede, and J. J. Steil. Teaching nullspace constraints in physical human-robot interaction using reservoir computing. In *International Conference on Robotics and Automation*, pages 1868–1875, 2012.

[110] D. A. Norman and T. Shallice. Attention to action: Willed and automatic control of behavior. In *Consciousness and self-regulation: advances in research and theory*, volume 4, pages 1–18. Springer, 1986.

[111] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Sequence of the most informative joints (SMIJ): A new representation for human skeletal action recognition. *Journal of Visual Communication and Image Representation*, 25(1):24–38, 2014.

[112] J. Pan, I. A. Sucan, S. Chitta, and D. Manocha. Real-time collision detection and distance computation on point cloud sensor data. In *International conference on robotics and automation*, pages 3593–3599, 2013.

[113] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013.

[114] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *International conference on humanoid robotics*, pages 91–98, 2008.

[115] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *International conference on humanoid robots*, pages 309–315, 2012.

[116] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[117] N. Perrin and P. Schlehuber-Caissier. Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems. *Systems & Control Letters*, 96:51–59, 2016.

[118] T. Petrič, A. Gams, L. Zlajpah, A. Ude, and J. Morimoto. Online approach for altering robot behaviors based on human in the loop coaching gestures. In *International Conference on Robotics and Automation*, pages 4770–4776, 2014.

[119] Y. Piao, K. Hayakawa, and J. Sato. Space–time invariants and video motion extraction from arbitrary viewpoints. In *International Conference on Pattern Recognition*, pages 56–59, 2002.

[120] Y. Piao, K. Hayakawa, and J. Sato. Space–time invariants for recognizing 3d motions from arbitrary viewpoints under perspective projection. In *International Conference on Image and Graphics*, pages 200–203, 2004.

[121] A. Psarrou, S. Gong, and M. Walter. Recognition of human gestures and behaviour based on motion trajectories. *Image and Vision Computing*, 20(5–6):349–358, 2002.

[122] S. Quinlan. Efficient distance computation between non-convex objects. In *International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[123] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[124] K. Ramirez-Amaro, M. Beetz, and G. Cheng. Understanding the intention of human activities through semantic perception: observation, understanding and execution on a humanoid robot. *Advanced robotics*, 29(5):345–362, 2015.

[125] C. Rao, M. Shah, and T. Syeda-Mahmood. Action recognition based on view invariant spatio-temporal analysis. In *ACM Multimedia*, 2003.

[126] C. Rao, A. Yilmaz, and M. Shah. View-invariant representation and recognition of actions. *International journal of computer vision*, 50(2):203–226, 2002.

[127] H. E. Rauch, C. T. Striebel, and F. Tung. Maximum likelihood estimates of linear dynamic systems. *Journal of the American Institute of Aeronautics and Astronautics*, 3(8):1445–1450, 1965.

[128] H. Ravichandar and A. P. Dani. Learning contracting nonlinear dynamics from human demonstrations for robot motion planning. In *Dynamics, systems and control conference*, 2015.

[129] M. A. Roa, M. J. Argus, D. Leidner, C. Borst, and G. Hirzinger. Power grasp planning for anthropomorphic robot hands. In *International Conference on Robotics and Automation*, pages 563–569, 2012.

[130] H. Sadeghian, L. Villani, M. Keshmiri, and B. Siciliano. Task-space control of robot manipulators with null-space compliance. *Transactions on robotics*, 30(2):493–506, 2014.

[131] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Transactions on acoustics, speech, and signal processing*, pages 43–49, 1978.

[132] P. Sanguansat. Multiple multidimensional sequence alignment using generalized dynamic time warping. *Transactions on mathematics*, 11(8):668–678, 2012.

[133] M. Saveriano, S. An, and D. Lee. Incremental kinesthetic teaching of end-effector and null-space motion primitives. In *International conference on robotics and automation*, pages 3570–3575, 2015.

[134] M. Saveriano, F. Hirt, and D. Lee. Human-aware motion reshaping using dynamical systems. *Pattern recognition letters*, 2017.

[135] M. Saveriano and D. Lee. Invariant representation for user independent motion recognition. In *International symposium on robot and human interactive communication*, pages 650–655, 2013.

[136] M. Saveriano and D. Lee. Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles. In *International conference on intelligent robots and systems*, pages 5380–5387, 2013.

[137] M. Saveriano and D. Lee. Distance based dynamical system modulation for reactive avoidance of moving obstacles. In *International conference on robotics and automation*, pages 5618–5623, 2014.

[138] M. Saveriano and D. Lee. Safe motion generation and online reshaping using dynamical systems. In *International conference on ubiquitous robots and ambient intelligence*, 2014.

[139] G. Schreiber, A. Stemmer, and R. Bischoff. The fast research interface for the kuka lightweight robot. In *International conference on robotics and automation, workshop on innovative robot control architectures for demanding (research) applications - How to modify and enhance commercial controllers*, pages 15–21, 2010.

[140] G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[141] D. Shah, P. Falco, M. Saveriano, and D. Lee. Encoding human actions with a frequency domain approach. In *International Conference on Intelligent Robots and Systems*, pages 5304–5311, 2016.

[142] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics - Modelling, Planning and Control*. Springer, 2009.

[143] J. Silvério, L. Rozo, S. Calinon, and D. G. Caldwell. Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems. In *International Conference on Intelligent Robots and Systems*, pages 464–470, 2015.

[144] E. A. Sisbot and R. Alami. A human-aware manipulation planner. *Transactions on Robotics*, 28(5):1045–1057, 2012.

[145] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *Transactions on Robotics*, 23(5):874–883, 2007.

[146] J. J. E. Slotine and W. Li. *Applied nonlinear control*. Prentice–Hall, 1991.

[147] R. Soloperto, M. Saveriano, and D. Lee. A bidirectional invariant representation of motion for gesture recognition and reproduction. In *International conference on robotics and automation*, pages 6146–6152, 2015.

[148] E. Sousa, W. Erlhagen, F. Ferreira, and E. Bicho. Off-line simulation inspires insight: A neurodynamics approach to efficient robot task learning. *Neural networks*, 72(12):123–139, 2015.

[149] W. Takano and Y. Nakamura. Real-time unsupervised segmentation of human whole-body motion and its application to humanoid robot acquisition of motion symbols. *Robotics and autonomous systems*, 75, Part B(1):260–272, 2016.

[150] M. Tenorth and M. Beetz. Knowrob – a knowledge processing infrastructure for cognition-enabled robots. *The international journal of robotics research*, 32(5):566–590, 2013.

[151] C. Towell, M. Howard, and S. Vijayakumar. Learning nullspace policies. In *International Conference on Intelligent Robots and Systems*, pages 241–248, 2010.

[152] J. Umlauft, D. Sieber, and S. Hirche. Dynamic movement primitives for cooperative manipulation and synchronized motions. In *International Conference on Robotics and Automation*, pages 766–771, 2014.

[153] A. Vazquez-Otero, J. Faigl, and A. P. Munuzuri. Path planning based on reaction-diffusion process. In *International conference on intelligent robots and systems*, pages 896–901, 2012.

[154] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space. In *International Conference on Machine Learning*, pages 288–293, 2000.

[155] M. Vochten, T. De Laet, and J. De Schutter. Comparison of rigid body motion trajectory descriptors for motion representation and recognition. In *International Conference on Robotics and Automation*, pages 3010–3017, 2015.

[156] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *International Conference on Humanoid Robots*, pages 189–195, 2013.

[157] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *Conference on Computer Vision and Pattern Recognition*, pages 1290–1297, 2012.

[158] P. Wang, W. Li, Z. Gao, C. Tang, J. Zhang, and P. Ogunbona. Convnets-based action recognition from depth maps through virtual cameras and pseudocoloring. In *International Conference on Multimedia*, pages 1119–1122, 2015.

[159] W. Wang and J. J. E. Slotine. On partial contraction analysis for coupled nonlinear oscillators. *Biological cybernetics*, 92(1):38–53, 2005.

[160] I. Weiss. Geometric invariants and object recognition. *International Journal of Computer Vision*, 10(3):207–231, 1993.

[161] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr. Cognitive agents - a procedural perspective relying on the predictability of object-action-complexes (oacs). *Robotics and Autonomous Systems*, 57(4):420–432, 2009.

[162] S. Wrede, C. Emmerich, R. Ricarda, A. Nordmann, A. Swadzba, and J. J. Steil. A user study on kinesthetic teaching of redundant robots in task and configuration space. *Journal of Human-Robot Interaction*, 2(1):56–81, 2013.

[163] S. Wu and Y. F. Li. On signature invariants for effective motion trajectory recognition. *International Journal of Robotic Research*, 27(8):895–917, 2008.

[164] S. Wu and Y. F. Li. Motion trajectory reproduction from generalized signature description. *Pattern recognition*, 43(1):204–221, 2010.

[165] L. Xia, C.-C. Chen, and J. K. Aggarwal. View invariant human action recognition using histograms of 3d joints. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–27, 2012.

[166] P. Yan, S. M. Khan, and M Shah. Learning 4d action feature models for arbitrary view action recognition. In *International Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2008.

[167] E. Yoshida and F. Kanehiro. Reactive robot motion using path replanning and deformation. In *International conference on robotics and automation*, pages 5456–5462, 2011.

[168] A. Zisserman and S. Maybank. A case against epipolar geometry. In *Applications of Invariance in Computer Vision*, volume 825 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 1994.

[169] R. Zoliner, M. Pardowitz, S. Knoop, and R. Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *International conference on robotics and automation*, pages 1535–1540, 2005.