A Dynamical System-based Approach to Modeling Stable Robot Control Policies via Imitation Learning

THÈSE N^{O} 5552 (2012)

PRÉSENTÉE LE 22 NOVEMBRE 2012

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE D'ALGORITHMES ET SYSTÈMES D'APPRENTISSAGE
PROGRAMME DOCTORAL EN SYSTÈMES DE PRODUCTION ET ROBOTIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Seyed Mohammad KHANSARI ZADEH

acceptée sur proposition du jury:

Prof. H. Bleuler, président du jury
Prof. A. Billard, directrice de thèse
Prof. M. Hasler, rapporteur
Prof. O. Khatib, rapporteur
Prof. S. Vijayakumar, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2012

Abstract

Despite tremendous advances in robotics, we are still amazed by the proficiency with which humans perform movements. Even new waves of robotic systems still rely heavily on hardcoded motions with a limited ability to react autonomously and robustly to a dynamically changing environment. This thesis focuses on providing possible mechanisms to push the level of adaptivity, reactivity, and robustness of robotic systems closer to human movements. Specifically, it aims at developing these mechanisms for a subclass of robot motions called *"reaching movements"*, i.e. movements in space stopping at a given target (also referred to as episodic motions, discrete motions, or point-to-point motions). These reaching movements can then be used as building blocks to form more advanced robot tasks. To achieve a high level of proficiency as described above, this thesis particularly seeks to derive control policies that: 1) *resemble* human motions, 2) *guarantee* the accomplishment of the task (if the target is reachable), and 3) can *instantly* adapt to changes in dynamic environments.

To avoid manually hardcoding robot motions, this thesis exploits the power of machine learning techniques and takes an Imitation Learning (IL) approach to build a generic model of robot movements from a few examples provided by an expert. To achieve the required level of robustness and reactivity, the perspective adopted in this thesis is that a reaching movement can be described with a nonlinear Dynamical System (DS). When building an estimate of DS from demonstrations, there are two key problems that need to be addressed: the problem of generating motions that resemble at best the demonstrations (the "how-to-imitate" problem), and most importantly, the problem of ensuring the accomplishment of the task, i.e. reaching the target (the "stability" problem). Although there are numerous well-established approaches in robotics that could answer each of these problems separately, tackling both problems simultaneously is challenging and has not been extensively studied yet.

This thesis first tackles the problem mentioned above by introducing an iterative method to build an estimate of *autonomous nonlinear DS* that are formulated as a mixture of Gaussian functions. This method minimizes the number of Gaussian functions required for achieving both *local asymptotic stability* at the target and accuracy in following demonstrations. We then extend this formulation and provide sufficient conditions to ensure *global asymptotic stability* of

autonomous DS at the target. In this approach, an estimation of the underlying DS is built by solving a constraint optimization problem, where the metric of accuracy and the stability conditions are formulated as the optimization objective and constraints, respectively. In addition to ensuring convergence of all motions to the target within the local or global stability regions, these approaches offer an *inherent* adaptability and robustness to changes in dynamic environments.

This thesis further extends the previous approaches and ensures *global asymptotic stability* of DS-based motions at the target *independently of the choice of the regression technique*. Therefore, it offers the possibility to choose the most appropriate regression technique based on the requirements of the task at hand without compromising DS stability. This approach also provides the possibility of *online learning* and using *a combination of two or more regression methods* to model more advanced robot tasks, and can be applied to estimate motions that are represented with both *autonomous* and *non-autonomous DS*.

Additionally, this thesis suggests a reformulation to modeling robot motions that allows encoding of a considerably wider set of tasks ranging from reaching movements to agile robot movements that require hitting a given target with a specific speed and direction. This approach is validated in the context of playing the challenging task of minigolf. Finally, the last part of this thesis proposes a DS-based approach to realtime obstacle avoidance. The presented approach provides a modulation that instantly modifies the robot's motion to avoid collision with multiple static and moving convex obstacles. This approach can be applied on all the techniques described above without affecting their adaptability, swiftness, or robustness.

The techniques that are developed in this thesis have been validated in simulation and on different robotic platforms including the humanoid robots HOAP-3 and iCub, and the robot arms KATANA, WAM, and LWR. Throughout this thesis we show that the DS-based approach to modeling robot discrete movements can offer a high level of adaptability, reactivity, and robustness almost effortlessly when interacting with dynamic environments.

Keywords: Reaching Movements, Dynamical Systems, Imitation Learning, Hitting Motions, Obstacle Avoidance, Stability, Adaptability, Robustness.

Résumé

En dépit des avancées importantes dans le domaine de la robotique, nous sommes toujours surpris de la maîtrise avec laquelle les humains accomplissent leurs gestes. Aujourd'hui encore, les nouveaux systèmes robotiques sont très dépendants de mouvements prédéterminés et leur capacité de réaction autonome est limitée et peu robuste face à un environnement dynamique. Cette thèse se penche sur les moyens de fournir des mécanismes pour augmenter le niveau d'adaptabilité, de réaction et robustesse de systèmes robotiques pour parvenir à se rapprocher des mouvements humains. Plus spécifiquement, elle vise à développer des mécanismes pour une sous-classe de mouvements robotiques appelée *"mouvement d'atteinte"*, par exemple des mouvements dans l'espace qui s'arrêtent à un objectif (aussi référencés comme mouvements épisodiques, mouvements discrets, ou mouvements de point-à-point). Ces mouvements d'atteinte peuvent ensuite être utilisés comme des composants pour construire des tâches robotiques plus avancées. Pour atteindre le niveau de maîtrise élevé décrit précédemment, cette thèse cherche à dériver des régles de contrôle qui: 1) *ressemblent* aux mouvements humains, 2) *garantissent* l'accomplissement de l'objectif (si celui-ci est atteignable), et 3) peuvent *instantanément* s'adapter à des environnements dynamiques.

Pour éviter de coder manuellement les mouvements du robot, cette thèse exploite la puissance des méthodes d'apprentissage et d'imitation pour construire un modèle générique de mouvements robotiques via des exemples donnés par un expert (humain). Pour garantir les niveaux voulus de robustesse et de réactivité, la perspective adoptée dans cette thèse est d'écrire les mouvements de point-à-point avec des systèmes dynamiques (SD) non-linéaires. Quand on construit une estimation d'un SD à partir de démonstrations, nous faisons face à deux problèmes clés qui doivent être abordés: générer des mouvements qui ressemblent le plus possible à ceux démontrés (le problème de "comment imiter"), et le plus important, assurer que l'objectif a bel et bien été rempli, par exemple arriver au but (le problème de "stabilité"). Bien qu'il y ait de nombreuses méthodes établies en robotique qui ont une réponse pour chaque question posée séparément, il n'y a pas de réponse unifiant les deux problèmes et c'est un défi qui n'a pas encore été étudié en détail.

Cette thèse aborde premièrement le problème mentionné précédemment en introduisant une méthode itérative pour construire une estimation d'un *système dynamique non-linéaire autonome* qui est exprimé par un ensemble de fonctions Gaussiennes. Cette méthode minimise le nombre de fonctions Gaussiennes requises pour garantir à la fois une *stabilité asymptotique locale* à l'objectif et le suivi des démonstrations fournies. Nous étendons ensuite ce formalisme et fournissons suffisamment de conditions pour assurer *une stabilité asymptotique globale* du SD autonome. Dans cette approche, une estimation du SD est construite via un problème d'optimisation avec contraintes, où la précision et les conditions de stabilité sont respectivement formulées comme l'objectif d'optimisation et de contraintes. En plus de pouvoir assurer la convergence stable de toutes les trajectoires qui mènent au but dans une région locale et globale, ces approches offrent une adaptation inhérente et robuste au changement qui prend place dans l'environnement.

Cette thèse présente une extension aux approches précédentes en assurant une *stabilité asymptotique globale* des mouvements des SD *indépendamment du choix des méthodes de régression*. Donc, il est aussi possible de choisir la méthode de régression la plus appropriée basée sur les exigences de la tâche requise, sans compromettre la stabilité du SD. En plus, cette approche permet un apprentissage en ligne, d'utiliser la *combinaison de plusieurs méthodes de régression* pour modéliser les mouvements d'atteinte d'un robot, et d'être appliquée pour estimer des mouvements représentés avec un *SD autonome et non-autonome*. De plus, cette thèse suggère une reformulation de la modélisation des mouvements robotiques qui permet d'encoder considérablement plus de comportements allant de mouvements d'atteinte à des mouvements agiles comme ceux qui sont nécessaires pour frapper un objet pour lui donner une vitesse et direction spécifiques. Nous validons cette approche dans le contexte du jeu de minigolf. Finalement, la dernière partie de cette thèse propose une méthode d'évitement d'obstacles basée sur des SD. Cette approche fournit une modulation qui permet immédiatement de modifier le mouvement du robot pour éviter toute collision avec une multitude d'objets statiques ou dynamiques (les objets dynamiques doivent être convexes). Cette méthode peut être appliquée sur toutes les techniques décrites précédemment sans avoir d'effet sur leurs adaptabilité, rapidité ou robustesse. Les techniques qui sont développées dans cette thèse ont été validées en simulation et sur différentes plateformes robotiques qui incluent les robots humanoïdes HOAP-3 et iCub, les bras robotisés KATANA, WAM et LWR. À travers cette thèse nous avons démontré que les méthodes basées sur les SD pour modéliser les mouvements discrets peuvent offrir un haut niveau d'adaptabilité, de réaction et de robustesse presque sans effort lors d'interactions avec un environnement dynamique.

Mots Clé: Mouvements d'atteinte, Systèmes dynamiques, Apprentissage par imitation, Évitement d'obstacles, Stabilité, Adaptabilité, Robustesse.

To my beloved Soha, who is the joy of my life
and an endless source of support and encouragement.

To my wonderful parents, to whom I owe
everything that I have and will have.

Acknowledgments

I would like to deeply thank Aude Billard for giving me the opportunity to pursue my PhD under her supervision on an interesting interdisciplinary topic that included robots, dynamics, control, machine learning, etc. This thesis would not have been possible without her continuous support, patience, advice, and availability. I can hardly express my gratitude in words to Aude for creating such an amazing lab that has been unique in many aspects. The availability of several advanced robots and equipments in our lab gave me an exceptional opportunity to work with many state-of-the-art platforms and to evaluate different aspects of my work in real world experiments. She provided me with countless possibilities to travel across the globe to present my work as well as to meet people in the robotics community. Thanks to her for all the time she spent to read and correct my articles and thesis and all her fruitful guidelines, without which this thesis would never have reached its current state.

I would like to thank my examiners Professors Khatib, Vijayakumar, and Hasler that have honored me by accepting to read my thesis. Your insightful comments were undoubtedly a great asset for this thesis.

I am thankful to EPFL and Switzerland for offering me to work in a great scientific atmosphere in one of the most gorgeous countries in the world. Many thanks to the European projects AMARSi (FP7-ICT-248311) and ROBOT@CWE (FP6-034002) for supporting my research.

I have enjoyed each moment of my PhD life thanks to the amazing atmosphere at our lab. Every working day was full of interesting scientific and non-scientific discussions with such an awesome multifaceted group of people. I would like to immensely thank Basilio Noris, with whom I had the opportunity to discuss life, music, tv, science, games, religion, and many other interesting topics. Basilio has been my C++ reference and endless source of cool and weird English words. My thanks also go to him for all his efforts to reading and correcting my articles and some chapters of my thesis. My thanks to his second half, Marion Hämmerli, for the nice discussions about cinemas, arts, Iran, etc.

I would like to thank Klas Kronander, my student at the beginning, my coauthor afterward, and my valuable friend throughout my PhD. Working with Klas was an amazing experience for me that yielded many fruitful discussions and results. Thanks to him and his wife Elin for organizing many amazing barbecues and sharing many joyful moments with me on the Swiss mountains.

I am immensely grateful to Eric Sauser for the RobotToolKit, an invaluable software package that helped me a lot to apply my method on real and simulated robots. Eric's passion for robots, his sense of humor, and his endless knowledge on C++ have been an inspiration and a pleasure for me. Many thanks to Daniel Grollman for all the nice and intuitive discussions we had, for reading and correcting my articles, for organizing many amazing social events, and for being my ski buddy on the mountains. My thanks to Martin Duvanel for developing the software for the stereo vision system, for his calm yet humorous personality, and for always gathering us for lunch on time. I would like to thank Jean-Baptiste Keller for all his efforts on repairing the robots and building new tools for my experiments, and for being my encyclopedia for doing things in Switzerland.

My thanks to Elena Gribovskaya, my cheerful and supportive friend, office-mate, and coauthor, with whom I shared many ups and downs of PhD life. Thanks to Sylvain Calinon for all the nice discussions we had, and his support at the beginning of my PhD. Thanks to Brenna D. Argall for her invaluable advice on different aspects of academic life, and for reading and correcting my articles. Many thanks to Seungsu Kim, my traveling buddy throughout the AMARSi project, my tennis partner, and my ski fellow.

My especial thanks to Sahar El-Khoury – my always smiling buddy – who has brought a new pulse of energy to the lab. My last year of PhD became even much more joyful thanks to all great moments, discussions, concerns, and laughs that we shared together. Many thanks also to her for being my goto person for any question about the French language.

My thank to Florent d'Hallouin for organizing many interesting events, for being my ski fellow, and for ordering coffee capsules on time. Thanks to David Koch, an incredibly nice person with whom I shared the experience of starting a doctoral study at LASA. Thanks to my recent office mates Nicolas Sommer and Lucia Pais for keeping the atmosphere of our office warm and joyful. My thanks also go to Nicolas for his always availability for social events, for being the source of Kinder Chocolate in the lab, and for all the nice French and Persian discussions that we had.

Thanks to Ajay Tanwani, the last PhD fellow during my PhD life, for being such a friendly and energetic labmate. My thank to Silvia Magrelli for nice Italian discussions and her incredibly delicious tiramisus. Thanks to Ashwini Shukla for all discussions about India and Cricket. Thanks to Luka Lukic for always seeking "something new, something wild". My thank to Guillaume de Chambrier for being an energetic new fellow with lots of sense of humor. And thanks to the rest of the lab Miao Li, Suphi Erden, Burak Zeydan, Keisuke Umezawa, Guillaume Pihen, and Otpal Vittoz. Discussing different aspects of life and work with you guys was indeed a unique experience for me.

I would like to thank our secretaries Karin Elsea and Joanna Erfani, and the secretaries of EDPR, Claire Chabanel and more recently Corinne Lebet, for taking care of the administrative aspects of my PhD.

I would like to thank a number of valuable friends outside my doctoral study, who has made living in Switzerland a very enjoyable and pleasant experience for me. My especial thanks to Arash, Asieh, Behzad, Farzin, Laleh, Mahshid, Mohsen, Mostafa, Parastoo, and Zohreh with whom I shared many great memories. Life in Lausanne couldn't be better thanks to your companionship, availability, and sense of humor. I would also like to thank Mohsen for all the unforgettable moments that we spent together at playing tennis, and for all the useful and distractive discussions that we had during and after the game. Also many thanks to the Iranian Students Association at EPFL for organizing many unforgettable Iranian cultural events in Lausanne.

I would have never reached the place where I am now without the endless support, love, and encouragement of my parents. I am immensely thankful for their confidence in me, which has allowed me to discover many aspects of life independently. Many thanks to my brothers Hasan, Mohsen, and Hamid, and my sisters in law Rahimeh and Zohreh for all their back up and the nice moments we have spent together.

Last, but far from least, I would like to deeply thank my lovely wife Soha for being an endless source of energy, enthusiasm, and encouragement. Our new life as a couple started right before the beginning of my PhD. It was an amazing feeling; opening a new chapter of life and study at the same time. I will definitely miss our nice daily discussions from home to EPFL and on the way back, the great feeling of discovering a great portion of the world together, our amazing ski, biking, and camping experiences in Switzerland, and many other invaluable moments. Infinite thanks to you for being my better half and my dearest companion.

Table of Contents

Chapter 1

Introduction

An inconvenience is only an adventure wrongly considered; an adventure is an inconvenience rightly considered.

Gilbert Keith Chesterton (1874-1936)

1.1 Motivations

From the moment you turn off the alarm clock in the morning till you turn it on again at night, you do numerous amounts of reaching movements without even noticing. For instance, imagine you are in a meeting. You are talking with your friend and at the same time reaching for an apple in the tray next to you. Whilst deep in your discussion and trying to reason out your argument, you start biting the apple. Your other friend — who is coincidentally a roboticist — does not enjoy the conversation, and instead is staring at your hand movement. She can clearly see that with every bite, your hand finds a different way to your mouth, though you may not even think about it. Even when the person next to you unintentionally bumps into your arm and you turn your head to see who has hit you, your hand can instantly adapt to these changes and reach your mouth effortlessly without pause.

Having robotic systems that exhibit the level of robustness and adaptability described above is essential, particularly if we envision to bring robots into our daily lives. Let us take another example. Imagine you are being served tea by a robot. As the robot is about to pour the boiling liquid in the cup you are holding, you sneeze. As a result of your sudden jolt, the cup is displaced and your hand is now in the way of the robot in place of the cup. It would be desirable that the robot to be able to react swiftly, and redirect its motion

to the cup while avoiding your hand. This is one of the many examples that emphasize the necessity for adaptive robotic systems that can react in the order of a second. In addition to the need of having adaptive robotic systems, it would be even more fascinating if we could achieve the described level of proficiency in an easy and intuitive way. In such a way, as it happens, that anybody could impart an instruction, education or command to a robot, even with no in-depth knowledge of engineering, of the robotic platform, or of the task at hand.

Devising a framework that can fulfil the combination of the above two requirements is nontrivial. For years classical control approaches have provided us with the tools to perform high precision tasks with industrial robots. However, these approaches rely heavily on hardcoded motions with a limited ability to adapt autonomously to a dynamically changing environment. Besides, they require a large amount of engineering knowledge and experience to efficiently put them to use. Other approaches, such as global path planning, are comparatively easier to use and can generate feasible trajectories that can fulfil the constraints of the task at hand (Diankov & Kuffner, 2007; Burns & Brock, 2005; Toussaint, 2009). Despite recent efforts at reducing the computational costs of such global searches for a feasible path, however, these methods cannot offer the reactivity sought to adapt to dynamic environments.

The main scope of this thesis is to devise a generic framework that fulfils the two important requirements described above. This framework is particularly developed for a subclass of robot motions called *"reaching movements"*. Modeling of reaching movements provides basic components, the so-called motion primitives (Schaal, 1999), which can be seen as a basis from which more advanced robot tasks can be formed[1]. To avoid manually hardcoding robot motions, this thesis exploits the power of machine learning techniques and takes an *Imitation Learning (IL)* approach to build a generic model of robot reaching movements from a few examples provided by the expert. To achieve the required level of robustness and reactivity, it formulates the encoding of reaching movements as a control law that is driven from *nonlinear dynamical systems*.

When modeling robot motions with nonlinear Dynamical Systems (DS), ensuring stability of the learned DS (from a set of demonstrations of the task) is a key requirement to provide a useful control policy. Hence, the major part of this thesis is concentrated on addressing the challenging problem of *"how to build a locally or globally stable estimate of nonlinear dynamical systems from a set of user demonstrations?"* The answer to this question is the core part of our devised framework. The remaining part of this thesis then focuses on extending the application area of this framework to modeling hitting movements and to performing obstacle avoidance in a dynamically changing environment.

[1] As an example, consider the standard "pick-and-place" task, which can be decomposed as follows: First reach to the item, then after grasping move to the target location, and finally return home after release.

1.2 Approach

As outlined in the previous section, this thesis aims at providing a generic framework to generate with ease robot control policies that exhibit a high level of proficiency and adaptivity. Specifically, our goal is to build an all-encompassing model of robot reaching movements that fulfils the following desiderata: 1) It should be estimated via imitation learning, 2) It should ensure accomplishment of the task as long as the target is reachable, 3) It should be robust to perturbations and adaptable to changes in a rather complex dynamic environments with several static and moving obstacles, and 4) It should perform the adaptation on-the-fly without any need to re-plan. The first criterion is essential as it provides an easy and intuitive means to program robots. Additionally, it allows generating motions that mimic human motions (which are more natural looking). The other three criteria are crucial since they increase the reliability of the approach, and are essential particularly when we have robotic systems that work in the close vicinity of humans. The approach devised in this thesis is founded on two main pillars: *Imitation Learning* and *Dynamical Systems*. Next, we describe how the field of robotics has evolved from the tedious hardcoding of robot motions to the appearance of learning-based approaches. Then we introduce imitation learning, a particular form of robot learning, and finally explain the emergence of dynamical system approaches to modeling robot motions.

1.2.1 Robot Learning

As outlined before, classical approaches to robot control rely on following preprogrammed motions with a limited level of adaptivity to changing environments. Manual programming of robot motions often requires a large amount of engineering knowledge about both the task and the robot and, above that, it can become particularly non-intuitive when dealing with high Degrees of Freedom (DoF) robotic systems or fulfilling requirements of complex tasks. Emergence of a new generation of robots that need to perform a wide variety of tasks in human daily lives stresses further more the importance of seeking alternative techniques as manual programming cannot be a reasonable option anymore. In response to these concerns, learning-based approaches appear as a promising route to automate the tedious manual programming phase by having a robot actually learn how to perform a desired task.

To date, there are two best known learning techniques that are actively used to generate robot movements, namely Reinforcement Learning (RL) (M. Waltz & Fu, 1965; Mendel & McLaren, 1970; Sutton & Barto, 1998) and Imitation Learning (IL) (Lozano-Perez, 1983; Segre & DeJong, 1985; Kuniyoshi et al., 1989). RL mainly focuses on a goal-oriented learning from the interaction with environments, whereas IL is mainly concerned with the ability to develop new skills from observing them being performed by another agent. Both approaches are founded on the learning mechanisms of biological systems. Throughout our

lives, we learn countless things through trial-and-error (a well-known example of RL) or by observing others' behavior, which provides us with a major source of knowledge about ourselves and our environment. The outcome of the learning process in both approaches is a states-to-actions map, also called *policy*. This policy can then be used by robots to interact with the world. RL builds such a map through exploration so as to maximize a numerical reward. In contrast, IL aims at deriving a policy that reproduces behaviors similar to those that are demonstrated. We will next describe IL-based approaches as it is the direction we take in this thesis.

1.2.2 Imitation Learning

Imitation Learning (also referred to programming by demonstrations, apprenticeship learning, or learning from demonstration) is one of the fundamental learning mechanisms in humans daily lives. Day in day out, we use IL so frequently that we seldom even notice. We employ IL for different purposes: from understanding a common social behavior in a small group to learning complex movements in sport games. In robotics, IL started attracting attention at the beginning of the 1980s (Lozano-Perez, 1983). Through the years, IL has been advocated as a powerful means to bootstrap robot learning (Kuniyoshi et al., 1989; Münch et al., 1994; Schaal, 1999). IL provides an intuitive way to transmit skills to robots without explicitly programming them. Moreover, it speeds up robot learning by reducing the search space of solutions (Billard et al., 2008). IL-based approaches have proved to be interesting alternative to classical control and planning methods in different applications such as locomotion (Ijspeert et al., 2002b), control of acrobatic helicopters (Coates et al., 2008), reaching movements (Calinon et al., 2007), etc. A more detailed overview of IL-based approaches is provided in Section 3.3.

IL can be performed at the symbolic or trajectory level. The former captures a high-level representation of a skill by decomposing it into a sequence of action-perception units. In contrast, the latter acts at a lower-level and encodes a nonlinear mapping between sensory and motor information. In both cases, the final goal is to perform a task as similar as possible to the examples provided by the teacher (also called expert or demonstrator). The notion of similarity is defined in terms of a *metric of imitation performance*. This metric provides a means to quantitatively express the user's intention during the demonstrations, and constitutes all features that remain invariant across them (Calinon, 2009). By using these features rather than only mimicking the teacher, IL-based approaches are able, to some extent, to generalize the task to unseen situations. Note that recent work on IL also considers the possibility that demonstrations are incorrect examples (Grollman & Billard, 2011). Thus, instead of maximizing the similarity of generated behaviors to demonstrations, they deliberately avoid repeating the human's mistakes.

When modeling robot motions via IL, it should be noted that inferring solely based on the user examples may not necessarily be sufficient to derive a useful control policy. User demonstrations are usually noisy, and the estimated model may not be accurate enough to satisfy a task's hard constraints. In many situations specifying the minimum accuracy required to fulfil the task's constraint is non-trivial, and quite often satisfying that requirement is much harder (if not impossible). Let us take the example of figure skating. In this sport, only a few combinations of the whole body movements are acceptable and leads to successful performance of intricate and challenging forms such as spins, jumps, footwork, etc. Even the slimmest change in the angle between the skate's blade and the ice could affect the performer's stability and cause her falling. For these kinds of tasks, it might be much easier to impose a task's hard constraints during the learning process to control the range of possible values that the learning parameters can take. Similarly, in the approach that is adopted in this thesis, the estimation of nonlinear DS through the direct use of existing techniques is not effective. These methods do not optimize under the constraint of making the system stable at the attractor, and thus, they are not guaranteed to result in motions that can reach the target. Hence this thesis develops an alternative statistical-based technique to build an estimate of nonlinear DS under strict stability conditions.

1.2.3 Dynamical Systems

Classical approaches to modeling robot motions rely on decomposing the task execution into two separate processes: *planning* and *execution* (Brock & Khatib, 1999). The former is used as a means to generate a feasible path that can satisfy the task's requirements, and the latter is designed so that it follows the generated feasible path as closely as possible. Hence these approaches consider any deviation from the desired path (due to perturbations or changes in environment) as the tracking error, and various control theories have been developed to efficiently suppress this error in terms of some objective functions. Despite the great success of these approaches in providing powerful robotic systems, particularly in factories, they are ill-suited for robotic systems that are aimed to work in the close vicinity of humans, and thus alternative techniques must be sought.

In robotics, DS-based approaches to motion generation have been proven to be interesting alternatives to classical methods as they offer a natural means to integrate planning and execution into one single unit (Kelso, 1995; Schaal et al., 2000; Billard & Hayes, 1999; Selverston, 1980). For instance when modeling robot reaching motions with DS, all possible solutions to reach the target are embedded into one single model. Such a model represents a global map which specifies *instantly* the correct direction for reaching the target, considering the current state of the robot, the target, and all the other objects in the robot's

working space. Clearly such models are more similar to human movements in that they can effortlessly adapt its motion to change in environments rather than stubbornly following the previous path. In other words, the main advantage of using DS-based formulation can be summarized as: "Modeling movements with DS allows having robotic systems that have inherent adaptivity to changes in a dynamic environment, and that can swiftly adopt a new path to reach the target". This advantage is the direct outcome of having a unified planning and execution unit.

Early approaches to DS-based motion generations include, for instance, Recurrent Neural Network (RNN) and its variants (B. A. Pearlmutter, 1995; Lin et al., 1995; Sudareshan & Condarcure, 1998; B. Pearlmutter, 1989; Pineda, 1987), Central Pattern Generators (CPG) for locomotion (Grillner, 1975; Raibert, 1986; Taga et al., 1991; Ijspeert et al., 1998), and Vector Integration To Endpoint (VITE) model for reaching movements (Bullock & Grossberg, 1988a; Gaudiano & Grossberg, 1992; Bullock et al., 1999). Schaal et al. (2000) were among the first groups to suggest the idea of using a programmable DS formulation that can be adjusted to different tasks. This idea was then further extended by Ijspeert et al. (2001), where they propose a method to build an estimate of nonlinear DS via IL. During the last decade, the IL-based approach to estimate DS model of robot motions has become increasingly popular as it offers a mechanism to exploit the advantages of both robot learning and DS modeling (Dixon & Khosla, 2004a; Kober, Mulling, et al., 2010; Ude et al., 2010; Hersch et al., 2008; Kulic et al., 2008; Calinon, D'halluin, et al., 2010). We will provide a more comprehensive review of these approaches in Section 3.3.2.

For the reasons outlined above, this thesis takes a DS approach and models robot reaching movements with *autonomous nonlinear multi-dimensional DS*. The choice of "autonomous" (also called time-invariant) DS is essential as it allows instant adaptation to perturbations without any need to re-plan. Encoding motions with multi-dimensional DS is also advantageous since it allows capturing correlation across all dimensions, which may be crucial for accurate reproduction. As none of the existing statistical tools could build a stable estimate of the above DS, this thesis presents three different statistical-based approaches to build an estimate of the adopted DS formulation under strict (local or global) stability constraint.

1.3 Contributions

The main contribution of this dissertation lies in providing a generic and unified framework capable of modeling various robot reaching movements, ranging from simple pick and place motions to agile striking movements, with both autonomous and non-autonomous nonlinear DS. The prominent features of this framework is the ability to generate robot reaching movements that 1) can be taught through a natural means that is accessible by naive users, 2) can

guarantee convergence to the target, 3) are inherently robust to perturbations, 4) can instantly perform collision avoidance in the presence of multiple static and moving obstacles, and 5) can adapt their motions on-the-fly to a dynamically changing environment. The novelties of the proposed framework revolve around the following four axes:

- Learning Algorithm

  The learning algorithms presented in this thesis can build an estimate of nonlinear multi-dimensional DS from a set of examples while ensuring its global or local asymptotic stability at the target. To date, existing DS-based approaches to encode robot motions rely either on some heuristics with the aim to build a locally stable estimate of nonlinear DS without any guarantee that such a model is feasible, or they depend on a (time-dependent) switching mechanism to ensure stability by switching from an unstable nonlinear DS to a stable linear DS. This is the first time that a statistical-based learning algorithm is suggested which can actually ensure global stability of nonlinear time-independent DS during the training phase.

- Robot Hitting Movements

  This thesis provides a novel scheme to perform hitting movements with a desired speed and direction at the target. The proposed approach differs from existing ones in that it decomposes learning of complex striking movements into two separate subtasks: 1) Learning how to hit the target, and 2) Learning to predict the proper hitting parameters, i.e. hitting angle and hitting speed. The basic hitting motion is modeled with an autonomous DS, and is learned with the algorithms described above. The hitting parameters are estimated based on a set of provided examples of good hitting parameters. In addition to all the aforementioned features due to DS modeling, the presented scheme offers two additional advantages: 1) It keeps the number of demonstrations small which is essential for an IL-based approach, 2) Given the appropriate adaptation, an acquired skill can be used to carry out a more complex task than the teacher is capable of demonstrating.

- Applicability to DS models that are formulated with Different Regression techniques

  There are numerous nonlinear regression techniques to estimate nonlinear DS (please refer to Section 2.2 for a review).

Each of these techniques has its own benefits and drawbacks which make their use very task-dependent. However, as outlined before, none of these techniques can be directly used to model DS-based robot motions because they do not ensure stability. This thesis develops a method to ensure stability of DS-based motions independently of the choice of the regression technique. Therefore, it allows adopting the most appropriate technique based on the requirements of the task at hand without compromising DS stability. This approach also provides the possibility of online learning and using a combination of two or more regression methods, which could be helpful to satisfy the requirements of more advanced robot tasks.

- Obstacle Avoidance

  This thesis also devises a DS-based obstacle avoidance approach that can be integrated into all the techniques described above. Hence it allows generating robot motions in more realistic situations where several static and moving objects may appear in the robot working space. The proposed approach has a level of reactivity similar to existing local obstacle avoidance methods, while it ensures convergence to the target proper to global obstacle avoidance techniques.

All the techniques presented in this thesis are validated through various simulation and real-world robot experiments. Moreover, their applicability to different robotic systems are verified on a number of robotic platforms with varying degrees of freedom including: the 4-DoF right arm of the humanoid robot HOAP-3, the 7-DoF right arm of the humanoid robot iCub, the 6-DoF robot arm KATANA, the 7-DoF robot arm WAM, and the 7-DoF robot arm LWR.

1.4 Thesis Outline

This thesis is composed of a number of chapters which are categorized according to the major topics they address. A brief overview of each chapter as well as their associated contributions are outlined below:

Chapter 2: Adopted Tools

This chapter provides a brief overview of the main tools that are taken from different research domains and extensively used in this thesis.

Chapter 3: Background Research

This chapter reviews the state of the art on different strategies that address the problem of motion generation for robot movements, with an emphasis on works specifically devised to generate discrete motions for manipulators.

Chapter 4: Learning Reaching Movements with DS

This chapter presents three methods to build a stable DS model of robot reaching movements from a set of demonstrations. The first method ensures local asymptotic stability of autonomous nonlinear DS that are formulated as a mixture of Gaussian functions. The second method precedes the previous one by ensuring global asymptotic stability at the target. The third approach provides a more generic framework by ensuring global asymptotic stability of both autonomous and non-autonomous DS that are formulated with any smooth regression technique.

Chapter 5: Learning Hitting Movements with DS

This chapter provides a reformulation to the DS model presented in Chapter 4, and substitutes the notion of (local or global) stability with (local or global) convergence. This reformulation allows encoding striking movements that require hitting the target at a desired direction and speed. This chapter further showcases the application of the presented approach in the context of playing minigolf in various challenging fields.

Chapter 6: DS-based Obstacle Avoidance

This chapter presents a DS-based approach to real-time obstacle avoidance. The presented method assumes the robot motion is driven by a continuous and differentiable DS in the absence of obstacle(s). Then it provides a modulation that instantly modifies the DS robot's motion to avoid collision with multiple static and moving convex obstacles. The modulation does not compromise the DS stability, and hence the convergence to the target is ensured. The proposed method can be applied to perform obstacle avoidance in Cartesian and joint spaces and is applicable to all the techniques presented in Chapters 4 and 5.

Chapter 7: Conclusion

This chapter concludes this thesis by summarizing its technical contributions, discussing its assumptions and limitations, and providing possible research extensions.

Appendices:

This thesis includes a number of appendices that provide supplementary information such as proofs of theorems, analytical derivations of some equations, further results and illustrations, etc.

1.5 Publication Note

Most of the material presented in Chapters 4 to 6 have been published in peer-reviewed conference proceedings and scientific journals. References to the related publications are provided at the beginning of each chapter. Furthermore, the videos of the robot experiments that are reported in this thesis are available online and can be downloaded from:

http://lasa.epfl.ch/

Chapter 2

Adopted Tools

> *If you want to make an apple pie from scratch, you must first create the universe.*
>
> Carl Sagan (1934-1996)

The techniques that are developed in this thesis are grounded on a number of tools taken from different research domains such as control engineering, machine learning, and mathematics. In this section we provide a brief overview of these tools, which is essential for understanding the techniques presented in this thesis.

As outlined before, throughout this thesis we take a dynamical system (DS) perspective to model robot motions. Thus in Section 2.1, we first describe a general description of DS and its main variants. Then we introduce a number of methods that are developed for stability analysis of nonlinear DS. These techniques provide a general guideline about how to verify stability of a given DS. We will exploit these techniques to ensure stability of our DS-based control policy in Chapters 4 to 6.

In addition to the DS representation, this thesis exploits the existing tools in machine learning to encode DS model of robot motions from a set of user demonstrations. A brief overview of these techniques is provided in Section 2.2. We will use these methods in Chapters 4 and 5 as the basic structure to encode robot motions.

The techniques that we present in this thesis deal in fact with a constrained optimization problem: "How to build an estimate of a DS under its strict stability constraint?". Thus, in Section 2.3, we provide an overview of standard constrained optimization techniques. We directly use these methods in Chapters 4 and 5 to find an optimal value of the DS model of robot motions under their strict stability conditions.

Finally as we choose to model robot motions at the kinematic level (i.e. the robot end-effector or joint angles), our approach relies on a low-level controller that converts kinematic variables into motor commands (e.g. force or torque). We delineate the control strategy that we adopt throughout this thesis in Section 2.4.

2.1 Dynamical Systems

Dynamical System (DS) is one of the fundamental tools that have been used by engineers to model a variety of physical systems ranging from electrical circuits to mechanical systems. DS provides an analytical description of the evolution of a system along time. Hence it can be used to predict the behavior of a system in the future, which is particularly important because it allows correcting the system behavior before it fails.

A continuous nonlinear DS can usually be represented by a set of nonlinear differential equations that are expressed in terms of time $t \in \mathbb{R}^+$, a d-dimensional vector of state variables $\boldsymbol{\xi} \in \mathbb{R}^d$, and an m-dimensional vector of input (or control) variables $\boldsymbol{u} \in \mathbb{R}^m$:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t, \boldsymbol{\xi}, \boldsymbol{u}), \qquad \boldsymbol{f} : \mathbb{R}^+ \times \mathbb{R}^d \times \mathbb{R}^m \mapsto \mathbb{R}^d \tag{2.1}$$

As it appears from their names, state variables $\boldsymbol{\xi}$ are a set of variables that entirely defines the state of a system, and input variables $\boldsymbol{u}$ are those that can be used to modify the evolution of $\boldsymbol{\xi}$. For example for a manipulator arm, it is very common to define the state and control variables as the set of all robot joint angles and motors, respectively. Eq. (2.1) is called the state equation. Without the explicit presence of $\boldsymbol{u}$, the state equation transforms into a so-called unforced state equation:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t, \boldsymbol{\xi}), \qquad \boldsymbol{f} : \mathbb{R}^+ \times \mathbb{R}^d \mapsto \mathbb{R}^d \tag{2.2}$$

It should be noted that the absence of $\boldsymbol{u}$ in Eq. (2.2) does not necessary mean that its value is zero. In fact, by defining $\boldsymbol{u}$ in terms of $\boldsymbol{\xi}$ and/or t, the control variable $\boldsymbol{u}$ can be eliminated yielding to an unforced state equation (this procedure, for example, is commonly performed when describing the closed-loop dynamics of a feedback control system).

Due to the explicit dependency of Eq. (2.2) on time, it is also called non-autonomous or time-variant. The state trajectory of a non-autonomous DS is dependent of the initial time. If $\boldsymbol{f}$ does not explicitly depend on time, it is said to be autonomous or time-invariant:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}), \qquad \boldsymbol{f} : \mathbb{R}^d \mapsto \mathbb{R}^d \tag{2.3}$$

In the major part of this thesis, we model robot motions with autonomous DS because they grant an inherent robustness to temporal perturbations which result from delays in execution of a task. Note that we distinguish between spatial and temporal perturbations as these result in different distortion of the estimated dynamics and hence require different means to tackle them. For example in robotic tasks, spatial perturbations would result from an imprecise localization of an object or from interacting with a dynamic environment where

either object(s) or the robot's arm may be moved by an external perturbation. Temporal perturbations arise typically when the robot is stopped momentarily due to the presence of an object, or due to safety issues (e.g. waiting till the operator has cleared the workspace).

When dealing with DS, analyzing the behavior of the system is essential in order to determine the limitation and strengths of the system. For a given DS, the first and the most important question about its behavior is whether it is stable. A stable DS benefits from some inherent features, which are crucial when modeling movement primitives. This thesis tackles the problem of ensuring the stability of a class of nonlinear DS and grounds this work on a number of well-established results on the stability of such systems, which we summarize in the next section for completeness of the report.

2.1.1 Stability Analysis of Dynamical Systems

Stability of DS is usually analyzed around some special points called *equilibrium points*:[1]

Definition 2.1 *A point $\boldsymbol{\xi}^*$ is called an equilibrium point if a DS that is initialized at $\boldsymbol{\xi}^*$, will remain there for all future time.*

Equilibrium points of an autonomous DS can be determined by computing the real roots of Eq. (2.3). At a given equilibrium point $\boldsymbol{\xi}^*$, the concept of stability is defined as follows:

Definition 2.2 *The equilibrium point $\boldsymbol{\xi} = \boldsymbol{\xi}^*$ of Eq. (2.3) is locally stable if for each $R > 0$, there is $r = r(R) > 0$ such that if the initial state $\|\boldsymbol{\xi}(0) - \boldsymbol{\xi}^*\| < r$, then the evolution of the system in time satisfies $\|\boldsymbol{\xi}(t) - \boldsymbol{\xi}^*\| < R$ for all $t \geq 0$.*

Definition 2.3 *The equilibrium point $\boldsymbol{\xi} = \boldsymbol{\xi}^*$ of Eq. (2.3) is locally asymptotically stable if it is stable and r can be chosen such that if $\|\boldsymbol{\xi}(0) - \boldsymbol{\xi}^*\| < r$, then it implies $\lim_{t\to\infty} \boldsymbol{\xi}(t) = \xi^*$.*

Definition 2.4 *The equilibrium point $\boldsymbol{\xi} = \boldsymbol{\xi}^*$ is said to be globally asymptotically stable if the asymptotic stability holds for any initial point, i.e. $\lim_{t\to\infty} \boldsymbol{\xi}(t) = \xi^*$, for all $\boldsymbol{\xi}(0) \in \mathbb{R}^d$.*

Studying stability of DS is a broad subject in the field of dynamics and control, which can generally be divided into linear and nonlinear systems. An autonomous linear DS has the following general form:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{A}\boldsymbol{\xi} + \boldsymbol{b} \tag{2.4}$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b} \in \mathbb{R}^d$ are a constant matrix and vector, respectively. Stability of linear dynamics has been studied extensively. A linear DS defined

[1] The presented materials in this section are adapted from (Slotine & Li, 1991; Khalil, 1996).

by Eq. (2.4) is globally asymptotically stable at its unique equilibrium point $\boldsymbol{\xi}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$ if and only if the real part of all eigenvalues of the matrix $\boldsymbol{A}$ are strictly negative.

In contrast to linear DS, stability analysis of nonlinear DS has been an active research topic and theoretical solutions exist only for particular cases. Studies over the past centuries have led to a number of tools, including Lyapunov methods (Salle & Lefschetz, 1961), contraction theories (Lohmiller & Slotine, 1998), describing functions (J. Hsu & Meyer, 1968), the passivity approach (Khalil, 1996), etc. Among these techniques, Lyapunov methods are the most common and general approaches for studying the stability of nonlinear DS (Slotine & Li, 1991). This thesis also exploits the two most famous Lyapunov methods for analyzing stability of nonlinear DS, namely Lyapunov indirect and direct methods.

Lyapunov's indirect method (also called Lyapunov's linearization method) provides a means to study the local stability of nonlinear DS in a small neighborhood around the equilibrium point $\boldsymbol{\xi}^*$ by approximating it with a linear DS. A continuously differentiable DS defined by Eq. (2.3) can be linearized around a point $\boldsymbol{\xi}^*$ by computing its Jacobian matrix:

$$\boldsymbol{A} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\xi}}\bigg|_{\boldsymbol{\xi}=\boldsymbol{\xi}^*} \tag{2.5}$$

Theorem 2.1 *An equilibrium point $\boldsymbol{\xi}^*$ of a continuously differentiable DS given by Eq. (2.3) is locally asymptotically stable if the real part of all eigenvalues of the matrix $\boldsymbol{A}$ given by Eq. (2.5) are strictly negative. This theorem is known as Lyapunov's indirect theorem.*

Although Lyapunov's indirect theorem provides an easy and quick means of verifying local stability of nonlinear DS, it could be quite limiting in that it does not define the extent of stability (i.e. the linearization is a good approximation of the nonlinear systems).

Lyapunov's direct method relaxes this constraint, and provides a more generic technique to analyze stability of nonlinear DS without any need to perform the linearization procedure. The direct method is the mathematical extension of a fundamental physical observation: "if the total energy of a mechanical system is continuously dissipated, then the system, whether linear or nonlinear, must eventually settled down to an equilibrium point". Therefore, stability analysis via using Lyapunov's direct method requires 1) finding a non-negative energy function $V(\boldsymbol{\xi}) \geq 0$ (also called Lyapunov function), and 2) verifying if it always decrease in a neighborhood around the equilibrium point $\boldsymbol{\xi}^*$. In other words:

Theorem 2.2 *Let $\boldsymbol{\xi}^*$ be an equilibrium point of the nonlinear DS given by Eq. (2.3), and $\Omega \subset \mathbb{R}^d$ be a domain containing $\boldsymbol{\xi}^*$. Let $V(\boldsymbol{\xi}) : \Omega \to \mathbb{R}$ be a scalar function with continuous first partial derivatives such as:*

$$V(\boldsymbol{\xi}^*) = 0 \tag{2.6a}$$
$$V(\boldsymbol{\xi}) > 0 \qquad \forall \boldsymbol{\xi} \in \Omega \setminus \boldsymbol{\xi}^* \tag{2.6b}$$
$$\dot{V}(\boldsymbol{\xi}^*) = 0 \tag{2.6c}$$
$$\dot{V}(\boldsymbol{\xi}) < 0 \qquad \forall \boldsymbol{\xi} \in \Omega \setminus \boldsymbol{\xi}^* \tag{2.6d}$$

then, $\boldsymbol{\xi}^*$ is locally asymptotically stable in Ω. In this case, the domain Ω is called stability domain (also referred to region of attraction or invariant set), and in fact should correspond to a level set of $V(\boldsymbol{\xi})$. This theorem is known as Lyapunov's stability theorem. If Ω expands to the whole state-space (i.e. $\Omega = \mathbb{R}^d$) and $V(\boldsymbol{\xi})$ is radially unbounded (i.e. $V(\boldsymbol{\xi}) \to \infty$ as $\|\boldsymbol{\xi}\| \to \infty$), then $\boldsymbol{\xi}^*$ is globally asymptotically stable.

2.2 Nonlinear Regression Techniques

Nonlinear regression techniques deal with the problem of building a continuous mapping function $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ based on a set of $\mathcal{T}$ training data points $\mathcal{D} : \{\boldsymbol{\xi}_{\mathcal{I}}^i, \boldsymbol{\xi}_{\mathcal{O}}^i\}_{i=1}^{\mathcal{T}}$, where $\boldsymbol{\xi}_{\mathcal{I}}^i \in \mathbb{R}^n$ and $\boldsymbol{\xi}_{\mathcal{O}}^i \in \mathbb{R}^m$ correspond to vectors of input and output variables, respectively. The regression function $\boldsymbol{f}$ is usually described in terms of a set of parameters $\boldsymbol{\theta}$, where an optimal value of $\boldsymbol{\theta}$ can be determined during the training. Once an estimate of $\boldsymbol{f}$ is obtained, then it can be used to predict the value of $\boldsymbol{\xi}_{\mathcal{O}}$ for a new input $\boldsymbol{\xi}_{\mathcal{I}}$:

$$\boldsymbol{\xi}_{\mathcal{O}}^* = \boldsymbol{f}(\boldsymbol{\xi}_{\mathcal{I}}^*; \boldsymbol{\theta}) \tag{2.7}$$

There are numerous regression techniques to build an estimate of $\boldsymbol{f}$, including linear regression techniques (C. Bishop, 2006; Kutner et al., 2005), Gaussian Mixture Regression (GMR) (McLachlan & Peel, 2000), Locally Weighted Projection Regression (LWPR) (Vijayakumar & Schaal, 2000; Schaal et al., 2002), Gaussian Process Regression (GPR) (Rasmussen & Williams, 2006), Support Vector Regression (SVR) (Smola & Schölkopf, 2004), various techniques based on Artificial Neural Networks (ANN) (C. M. Bishop, 1995; Lee, 2004; Jordan & Rumelhart, 1992), etc. Each of these techniques has its own pros and cons which make their use very task-dependent.

The above approaches have been widely used in robotics for various applications such as encoding robot motions (Kulic et al., 2008; Muehlig et al., 2009; Yamane et al., 2004; Ijspeert et al., 2002b; Calinon et al., 2007; Hersch et al., 2008), learning robots model (Jordan & Rumelhart, 1992; Sigaud et al., 2011; Peters & Schaal, 2008; Vijayakumar et al., 2005; Nguyen-Tuong et al., 2008; Butz et al., 2007; Jaeger et al., 2007), recognition of gestures (Ardizzone et al., 2000; Waldherr et al., 2000; Zollner et al., 2002), face detection (Osuna et al., 1997; Li et al., 2000; Bartlett et al., 2003), etc.

It should be noted that a large body of work in control engineering is also devoted to the problem of modeling DS from experimental data (Gevers, 2006; Zadeh, 1956; Aström & Bohlin, 1965; Ljung, 1999; Ho & Kalman, 1966; Söderström & Stoica, 1989). This domain is usually referred to as *system identification*, and in general terms, is composed of three steps: 1) Collecting a data set through an identification experiment which consists of exciting the system using some input signals (such as step, impulse, random, sinusoid, etc.), and then observing (recording) the behavior of the system over a time interval, 2) Determining an appropriate form of the model for the system at hand. In this step, some insight from laws of physics is usually taken into account in order to determine the structure of the model and its unknown parameters, and 3) Using some statistical-based techniques to estimate the unknown parameters of the model using the obtained data set in step one.

The above procedure is also known as parametric system identification since the structure of the underlying system is known, and only a set of unknown parameters need to be identified. Alternatively, one can take a nonparametric system identification approach when there is no knowledge about the system's structure. Nonparametric approaches determines the output of the system at a point $\boldsymbol{\xi}$ based on a weighted average of a neighborhood around that point (Ljung, 2010). The problem that we consider in this thesis falls within the scope of nonparametric system identification as we have no knowledge about the structure of the DS that the observed data are generated from.

Aside from control engineering, the problem of building a mathematical model of (dynamical) systems has been an active research topic in many other scientific communities such as econometrics, statistics, and so on (which some of them appear to be converging to each other). Reviewing all these works is not possible in this thesis, and we refer interested readers to (Ljung, 2010; Aström & Eykhoff, 1971). A nice history of the evolution of system identification algorithms in different disciplines is also provided in (Gevers, 2006).

As outlined before, in this thesis we take a machine learning perspective to model robot motions. Next, we give a brief overview of the four regression techniques that are used in this thesis, namely GMR, GPR, SVR, and LWPR. Note that in this section we only present the basic equations describing these approaches. We will provide a more in-depth discussion and comparison between them in Chapter 4.

2.2.1 Gaussian Mixture Regression (GMR)

Gaussian Mixture Regression (GMR) is a nonlinear regression technique that works on the joint probability $\mathcal{P}([\boldsymbol{\xi}_\mathcal{I}; \boldsymbol{\xi}_\mathcal{O}])$ between input and output variables[2]. The joint probability is formed by superposition of K linear Gaussian functions:

[2]Note that we use the expression $[\boldsymbol{\xi}_\mathcal{I}; \boldsymbol{\xi}_\mathcal{O}]$ to vertically concatenate the two column vectors $\boldsymbol{\xi}_\mathcal{I}$ and $\boldsymbol{\xi}_\mathcal{O}$. The resulting vector $[\boldsymbol{\xi}_\mathcal{I}; \boldsymbol{\xi}_\mathcal{O}]$ has the dimension $n + m$.

$$\mathcal{P}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]) = \sum_{k=1}^{K} \pi^k \mathcal{N}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]|\boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k) \tag{2.8}$$

where π^k, $\boldsymbol{\mu}^k$ and $\boldsymbol{\Sigma}^k$ respectively are the prior, mean and covariance matrix of the k-th Gaussian function $\mathcal{N}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]|\boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k)$ that is described by:

$$\mathcal{N}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}];\boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k) = \frac{1}{\sqrt{(2\pi)^{n+m}|\boldsymbol{\Sigma}^k|}} e^{-\frac{1}{2}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]-\boldsymbol{\mu}^k)^T(\boldsymbol{\Sigma}^k)^{-1}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]-\boldsymbol{\mu}^k)} \tag{2.9}$$

where $(.)^T$ denotes the transpose. From different perspective, Eq. (2.8) can be rendered as:

$$\mathcal{P}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]) = \sum_{k=1}^{K} \mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]|k) \tag{2.10}$$

in which $\mathcal{P}(k) = \pi^k$ is the probability of picking the k-th component and $\mathcal{P}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]|k)$ stands for the probability the datapoint $[\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}]$ belongs to this component. In mixture modeling, the unknown parameters of the joint distribution are the priors π^k, the means $\boldsymbol{\mu}^k$ and the covariance matrices $\boldsymbol{\Sigma}^k$ of the $k = 1..K$ Gaussian functions (i.e. $\theta^k = \{\pi^k, \boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k\}$ and $\boldsymbol{\theta} = \{\theta^1..\theta^K\}$), which can be estimated by using an Expectation-Maximization (EM) algorithm (Dempster & Rubin, 1977). EM proceeds by maximizing the likelihood that the complete model represents the training data well. Given the joint distribution $\mathcal{P}([\boldsymbol{\xi}_{\mathcal{I}};\boldsymbol{\xi}_{\mathcal{O}}])$ and a query point $\boldsymbol{\xi}^*_{\mathcal{I}}$, the GMR process consists of taking the posterior mean estimate of the conditional distribution:

$$\boldsymbol{\xi}^*_{\mathcal{O}} = \boldsymbol{f}(\boldsymbol{\xi}^*_{\mathcal{I}};\boldsymbol{\theta}) = \mathbb{E}[\mathcal{P}(\boldsymbol{\xi}^*_{\mathcal{O}}|\boldsymbol{\xi}^*_{\mathcal{I}},\boldsymbol{\theta})] \tag{2.11}$$

By defining the components of the mean and the covariance matrix of a Gaussian k as:

$$\boldsymbol{\mu}^k = \begin{pmatrix} \boldsymbol{\mu}^k_{\mathcal{I}} \\ \boldsymbol{\mu}^k_{\mathcal{O}} \end{pmatrix} \quad \& \quad \boldsymbol{\Sigma}^k = \begin{pmatrix} \boldsymbol{\Sigma}^k_{\mathcal{I}} & \boldsymbol{\Sigma}^k_{\mathcal{I}\mathcal{O}} \\ \boldsymbol{\Sigma}^k_{\mathcal{O}\mathcal{I}} & \boldsymbol{\Sigma}^k_{\mathcal{O}} \end{pmatrix} \tag{2.12}$$

the expected distribution of $\boldsymbol{\xi}^*_{\mathcal{O}}$ can be estimated as:

$$\boldsymbol{\xi}^*_{\mathcal{O}} = \sum_{k=1}^{K} h(\boldsymbol{\xi}^*_{\mathcal{I}};\boldsymbol{\theta}^k)\left(\boldsymbol{\Sigma}^k_{\mathcal{O}\mathcal{I}}(\boldsymbol{\Sigma}^k_{\mathcal{I}})^{-1}(\boldsymbol{\xi}^*_{\mathcal{I}} - \boldsymbol{\mu}^k_{\mathcal{I}}) + \boldsymbol{\mu}^k_{\mathcal{O}}\right) \tag{2.13}$$

where

$$h(\boldsymbol{\xi}^*_{\mathcal{I}};\boldsymbol{\theta}^k) = \frac{\mathcal{P}(k)\mathcal{P}(\boldsymbol{\xi}^*_{\mathcal{I}}|k)}{\sum_{i=1}^{K}\mathcal{P}(i)\mathcal{P}(\boldsymbol{\xi}^*_{\mathcal{I}}|i)} = \frac{\pi^k\mathcal{N}(\boldsymbol{\xi}^*_{\mathcal{I}}|\boldsymbol{\mu}^k_{\mathcal{I}},\boldsymbol{\Sigma}^k_{\mathcal{I}})}{\sum_{i=1}^{K}\pi^i\mathcal{N}(\boldsymbol{\xi}^*_{\mathcal{I}}|\boldsymbol{\mu}^i_{\mathcal{I}},\boldsymbol{\Sigma}^i_{\mathcal{I}})} \tag{2.14}$$

Fig. 2.1 illustrates an example of using GMR to build an estimate of $\boldsymbol{f}$ from a set of noisy samples using 3 Gaussian functions. For illustrative purpose, a uni-dimensional input and output variables are considered in this example.

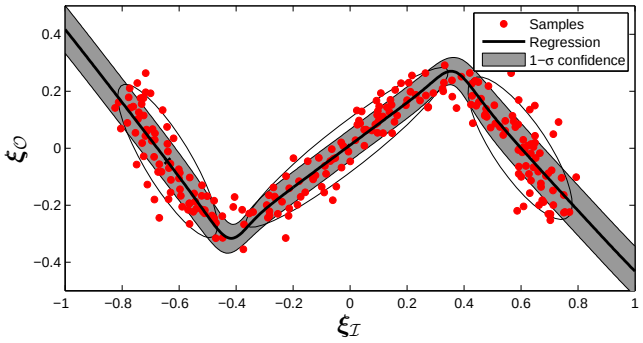


Figure 2.1: An examples of using GMR to regress a set of one-input one-output datapoints. In this graph, the ellipses and crosses represents the $3-\Sigma$ and centers of the Gaussian functions. The grey area represents the predictive confidence by one standard deviation.

2.2.2 Gaussian Process Regression (GPR)

Gaussian Process Regression (GPR) provides an estimate of the function $\boldsymbol{f}$ by assuming it is a Gaussian process. This assumption implies any set of samples from this function have a joint Gaussian distribution. Let us denote the set of $\mathcal{T}$ training datapoints with their corresponding uni-dimensional function values by $\boldsymbol{\Xi}_{\mathcal{I}} = \{\boldsymbol{\xi}_{\mathcal{I}}^i\}_{i=1}^{\mathcal{T}}$ and $\boldsymbol{\xi}_{\mathcal{O}} = \{\xi_{\mathcal{O}}^i\}_{i=1}^{\mathcal{T}}$, respectively. Note that $\boldsymbol{\Xi}_{\mathcal{I}}$ is an $n \times \mathcal{T}$ matrix whose i-th column corresponds to the i-th datapoint, and $\boldsymbol{\xi}_{\mathcal{O}}$ is also a row vector composed of $\mathcal{T}$ components. For any test point $\boldsymbol{\xi}_{\mathcal{I}}^*$, by conditioning the multivariate Gaussian distribution on the training data we obtain the GPR:

$$f_j(\boldsymbol{\xi}_{\mathcal{I}}^*)|\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\xi}_{\mathcal{O}} \sim \mathcal{N}\left(\mu(\boldsymbol{\xi}_{\mathcal{I}}^*), \Sigma(\boldsymbol{\xi}_{\mathcal{I}}^*)\right) \qquad \forall j \in 1..m$$

with the estimate $\mu(\boldsymbol{\xi}_{\mathcal{I}}^*)$ and the predictive variance $\Sigma(\boldsymbol{\xi}_{\mathcal{I}}^*)$ are given by:

$$\mu(\boldsymbol{\xi}_{\mathcal{I}}^*) = \boldsymbol{K}(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\Xi}_{\mathcal{I}})(\boldsymbol{K}(\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{\mathcal{I}}) + \sigma_n \boldsymbol{I})^{-1} \boldsymbol{\xi}_{\mathcal{O}}^T \tag{2.15a}$$

$$\Sigma(\boldsymbol{\xi}_{\mathcal{I}}^*) = \boldsymbol{K}(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\xi}_{\mathcal{I}}^*) - \boldsymbol{K}(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\Xi}_{\mathcal{I}})(\boldsymbol{K}(\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{\mathcal{I}}))^{-1} \boldsymbol{K}(\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\xi}_{\mathcal{I}}^*) \tag{2.15b}$$

The symmetric matrices $\boldsymbol{K}$ above represent the evaluation of the GP covariance function across the specified variables. We use a squared exponential with different length scales for the different dimensions in input space:

$$\boldsymbol{K}(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\Xi}_{\mathcal{I}}) = \begin{bmatrix} k(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\xi}_{\mathcal{I}}^1) & \dots & k(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\xi}_{\mathcal{I}}^{\mathcal{T}}) \end{bmatrix} \tag{2.16a}$$

$$\boldsymbol{K}(\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{\mathcal{I}}) = \begin{bmatrix} k(\boldsymbol{\xi}_{\mathcal{I}}^1, \boldsymbol{\xi}_{\mathcal{I}}^1) & \dots & k(\boldsymbol{\xi}_{\mathcal{I}}^1, \boldsymbol{\xi}_{\mathcal{I}}^{\mathcal{T}}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{\xi}_{\mathcal{I}}^{\mathcal{T}}, \boldsymbol{\xi}_{\mathcal{I}}^1) & \dots & k(\boldsymbol{\xi}_{\mathcal{I}}^{\mathcal{T}}, \boldsymbol{\xi}_{\mathcal{I}}^{\mathcal{T}}) \end{bmatrix} \tag{2.16b}$$

and $\boldsymbol{K}(\boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\xi}_{\mathcal{I}}^*) = \boldsymbol{K}(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\Xi}_{\mathcal{I}})^T$. The component $k(\boldsymbol{\xi}, \boldsymbol{\xi}')$ are usually computed using a squared exponential function:

$$k(\boldsymbol{\xi}, \boldsymbol{\xi}') = \sigma e^{-(\boldsymbol{\xi}-\boldsymbol{\xi}')^T \boldsymbol{L}(\boldsymbol{\xi}-\boldsymbol{\xi}')} \tag{2.17}$$

where $\boldsymbol{L}$ is an $n \times n$ diagonal matrix of length scales, and σ is the signal variance.

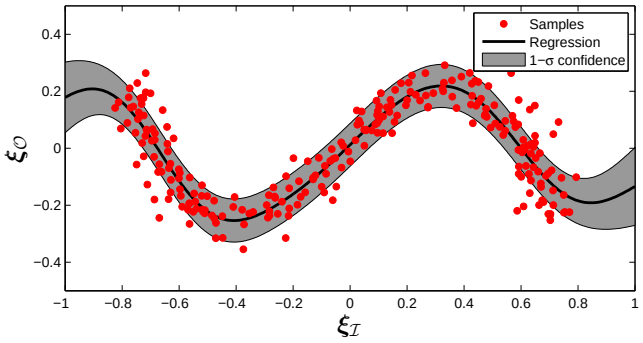


Figure 2.2: An examples of using GPR to regress a set of one-input one-output datapoints. The grey area represents the predictive confidence by one standard deviation.

The matrix $\boldsymbol{L}$ together with σ are called hyper-parameters and are denoted with $\boldsymbol{\theta}$. An optimal value of these parameters can be obtained by optimizing them for the maximum likelihood of the training set by using, for instance, a conjugate-gradient based search algorithm available in GPML[3]. Note that the general GPR formulation is only applicable to Multi-Input Single-Output (MISO) datasets. Thus, for data sets with multiple output, one needs to train a separate GPR model for each output dimension:

$$\boldsymbol{\xi}^*_{\mathcal{O}} = \boldsymbol{f}(\boldsymbol{\xi}_{\mathcal{I}}; \boldsymbol{\theta}, \boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{\mathcal{O}}) = \begin{bmatrix} \mu_1(\boldsymbol{\xi}^*_{\mathcal{I}}; \boldsymbol{\theta}_1, \boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{1,\mathcal{O}}) \\ \vdots \\ \mu_m(\boldsymbol{\xi}^*_{\mathcal{I}}; \boldsymbol{\theta}_m, \boldsymbol{\Xi}_{\mathcal{I}}, \boldsymbol{\Xi}_{m,\mathcal{O}}) \end{bmatrix} \tag{2.18}$$

where $\boldsymbol{\Xi}_{i,\mathcal{O}}$ corresponds to the i-th row of the matrix of output values. Fig. 2.2 illustrates an example of using GPR to build an estimate of $\boldsymbol{f}$ using the same data set as the one used for GMR.

2.2.3 Support Vector Regression (SVR)

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) to regression problems. It assumes the function $\boldsymbol{f}$ can be parameterized as:

$$f(\boldsymbol{\xi}^*_{\mathcal{I}}; \boldsymbol{\theta}) = \boldsymbol{\phi}(\boldsymbol{\xi}^*_{\mathcal{I}})^T \boldsymbol{\theta} \tag{2.19}$$

where $\boldsymbol{\phi}(\boldsymbol{\xi}^*_{\mathcal{I}})$ denotes a fixed feature-space transformation. The weighting vector $\boldsymbol{\theta}$ is the model parameters that can be determined through a constrained optimization problem on the training data set. The vector $\boldsymbol{\theta}$ is sparse and its non-zero elements corresponds to a fraction of training data that contribute to predictions. These data points are called support vectors.

There are two most common formulations of SVR, namely ϵ-SVR and ν-SVR. The former utilizes an ϵ-sensitive error function, and thus tolerates a maximum estimation error of ϵ in the predictions. In this approach, the support vectors lie

[3]GPML is a Matlab toolbox for GPR, written by C.E. Rasmussen and H. Nickisch.

on the boundary or outside of the ϵ-tube around the regression curve. Although this approach does not directly allow to constraint the number of support vectors n_f, one could generally expect n_f should decrease as ϵ increases. Contrary to ϵ-SVR, the alternative formulation given by ν-SVR provides a more intuitive way to set the number of support vectors. In this approach, the parameter ν bounds the fraction of points lying outside the tube. For both formulations, solving the optimization leads to:

$$\boldsymbol{\theta} = \sum_{i=1}^{n_f} \beta_i \boldsymbol{\phi}(\boldsymbol{\xi}^i) \tag{2.20}$$

where β is a vector of coefficients that is determined from the optimization. Substituting Eq. (2.20) into Eq. (2.19) yields:

$$\begin{aligned} \xi_{\mathcal{O}}^* = f(\boldsymbol{\xi}_{\mathcal{I}}^*; \boldsymbol{\theta}) &= \sum_{i=1}^{n_f} \beta_i \left(\boldsymbol{\phi}(\boldsymbol{\xi}_{\mathcal{I}}^*)\right)^T \boldsymbol{\phi}(\boldsymbol{\xi}_{\mathcal{I}}^i) \\ &= \sum_{i=1}^{n_f} \beta_i k(\boldsymbol{\xi}_{\mathcal{I}}^*, \boldsymbol{\xi}_{\mathcal{I}}^i) \end{aligned} \tag{2.21}$$

Similarly to GPR, SVR also expresses the predicted value in terms of a kernel function $k(\boldsymbol{\xi}, \boldsymbol{\xi}')$. The squared exponential function given by Eq. (2.17) is also commonly used in SVR, with the difference that the kernel width $\boldsymbol{L}$ should be now preset by the user. Note that SVR is also a MISO regression technique. Thus for multi-output datapoints, one should train a separate model for each output dimension. Fig. 2.3 illustrates examples of using ϵ-SVR and ν-SVR to build an estimate of $\boldsymbol{f}$ using the same data set as the one used for GMR.

2.2.4 Locally Weighted Projection Regression (LWPR)

Locally Weighted Projection Regression (LWPR) is an incremental regression technique that provides an estimate of $\boldsymbol{f}$ in terms of the output from a set of local regions, called Receptive Fields (RF). LWPR defines each RF $\omega(\boldsymbol{\xi})$ with a Gaussian function:

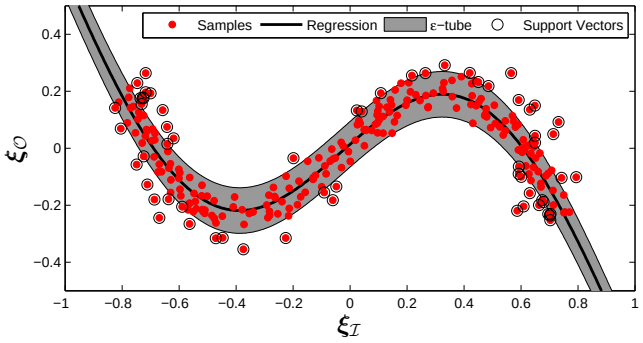
$$\omega(\boldsymbol{\xi}) = e^{-(\boldsymbol{\xi}-\boldsymbol{\mu})^T \boldsymbol{W} (\boldsymbol{\xi}-\boldsymbol{\mu})} \tag{2.22}$$

where $\boldsymbol{\mu}$ is the center of RF, and $\boldsymbol{W}$ is a positive semi-definite distance metric which determines the influence region of the RF. Prediction in LWPR starts by evaluating the output of each RF, which is described by a linear function with the parameters $\boldsymbol{A}$ and $\boldsymbol{a}$:

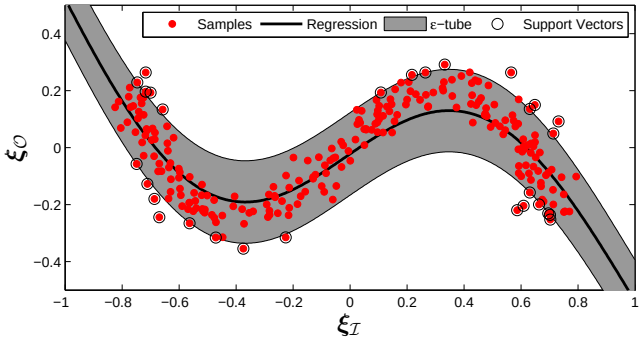
$$\boldsymbol{r}(\boldsymbol{\xi}_{\mathcal{I}}^*) = \boldsymbol{A}\boldsymbol{\xi}_{\mathcal{I}}^* + \boldsymbol{a}_0 \tag{2.23}$$

The final prediction is computed as a nonlinear weighted sum of the output of all RFs:

$$\boldsymbol{\xi}_{\mathcal{O}}^* = \boldsymbol{f}(\boldsymbol{\xi}_{\mathcal{I}}^*) = \frac{1}{\sum_{i=1}^{n_r} \omega^i(\boldsymbol{\xi}_{\mathcal{I}}^*)} \sum_{k=1}^{n_r} \omega^k(\boldsymbol{\xi}_{\mathcal{I}}^*) \boldsymbol{r}^k(\boldsymbol{\xi}_{\mathcal{I}}^*) \tag{2.24}$$



(a) ϵ-SVR



(b) ν-SVR

Figure 2.3: Examples of using SVR to regress a set of one-input one-output datapoints using **(a):** ϵ-SVR and **(b):** ν-SVR. In total 62 support vectors were obtained in ϵ-SVR, corresponding to 25% of the total number of training datapoints. By setting $\nu = 0.1$, the number support vectors decreases, resulting in having a wider ϵ-tube.

LWPR automatically updates all the parameters of RFs except the centers as a new training data arrives. It also controls the addition of new RFs or removal of unnecessary RFs through a threshold parameter. Fig. 2.4 illustrates an example of using LWPR to build an estimate of $\boldsymbol{f}$ using the same data set as the one used for GMR.

2.3 Optimization

When describing robot motions with DS, as we will highlight later on in Chapter 4, not only should the estimated DS be accurate, but it must also be stable. However, none of the regression techniques that are described in Section 2.2 considers the stability of DS during the estimation. Therefore, in this thesis we use optimization as a means to build an estimate of the DS motion under strict stability constraints.

The desire to optimize decisions arises naturally when there are several ways of doing a task. Optimization algorithms provide us with some mathematical tools to achieve this desired optimality whenever the task can be described quantitatively. An optimization problem is usually described as a minimization

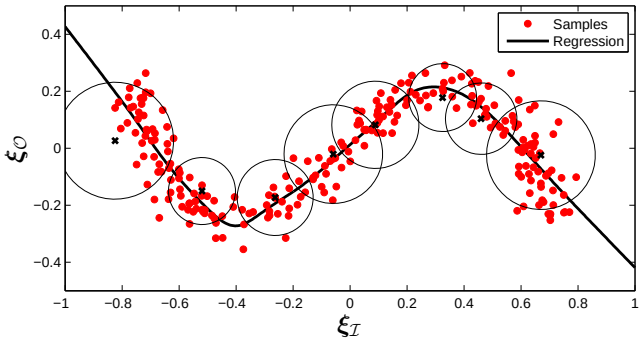


Figure 2.4: An examples of using LWPR to regress a set of one-input one-output datapoints. In this graph, the circles represents the RFs.

problem and has the general form:[4]

$$\begin{aligned} &\text{minimize} \quad J(\boldsymbol{\theta}) \\ &\text{subject to} \\ &\qquad \mathcal{L}_i(\boldsymbol{\theta}) = 0 \qquad i = 1..n_l \\ &\qquad \mathcal{C}_j(\boldsymbol{\theta}) \leq 0 \qquad j = 1..n_c \end{aligned} \tag{2.25}$$

where $\boldsymbol{\theta}$ is a p-dimensional vector of optimization variable (also called optimization parameter), the function $J(\boldsymbol{\theta})$ is the objective function, and the functions $\mathcal{L}_i$ and $\mathcal{C}_j$ are the optimization equality and inequality constraints. The ideal goal in minimization is to find an optimal value $\boldsymbol{\theta}^*$ such that $J(\boldsymbol{\theta}^*)$ has the smallest value in the feasible set (i.e. the set of points that satisfies the optimization constraints).

Since the late 1940s, a large effort has gone into developing algorithms for solving various classes of optimization problem. Generally, optimization problems can be split into two categories: convex and non-convex optimizations. The former happens when both the objective and the constraints are convex functions. In this case, if $\boldsymbol{\theta}^*$ exists, then it is the global optimum (Boyd & Vandenberghe, 2004). An optimization problem is non-convex if it has at least one non-convex element. Solving non-convex problems are significantly harder than convex problems. This difficulty is mainly due to the possible existence of several *local minima*. A point $\boldsymbol{\theta}^*$ is called local minimum if it only minimizes the objective among feasible points near it, but it does not have the lowest objective value amongst all feasible points.

An optimal value of a nonlinear non-convex problems can be obtained using a number of optimization algorithms. Local approaches such as active-set strategies, sequential quadratic programming, interior-point methods aim at finding a local minima (Bazaraa et al., 2006; Bonnans & Gilbert, 2006). These approaches can be fast and can handle large scale problems, but they have one main shortcoming: they rely on an initial guess for the optimization parameter.

[4]Note that any maximization problem can be formulated as a minimization problem by multiplying its objective function with -1.

The choice of initial guess is very critical as a bad initial point may lead to a rather poor local minimum. In contrast to this concern, other approaches such as genetic algorithms (Goldberg, 1989), particle swarm optimization (Eberhart et al., 2001), and simulated annealing (Davis, 1987) were suggested in order to find a comparatively better local minimum, ideally the global one. However, the computational complexity of these approaches grows significantly with the problem size, and thus their application is mostly limited to problems with a small number of variables.

In the most part of this thesis, we take an optimization perspective to build a (locally) optimal estimate of robot motions. We solely use one of the state-of-the-art local optimization techniques, namely the interior-point method, to find the optimal parameter. Here we give a brief overview of this method. More detail information about these approaches can be found in (Bonnans & Gilbert, 2006).

2.3.1 Interior-Point Method

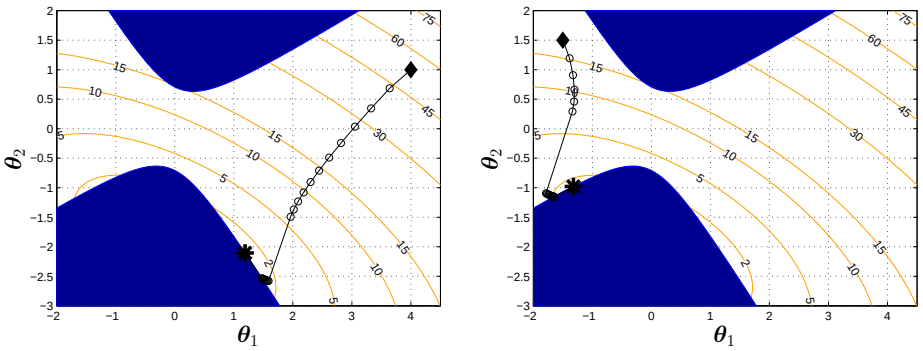
The main goal in the interior-point methods is to transform a rather difficult problem into a sequence of simpler subproblems which converge to the solution of the original problem either in a finite number of steps or in the limit. These methods exploit the so-called barrier functions to approximate a constrained optimization problem with an unconstrained problem. The latter is a much simpler problem that can be more efficiently solved with Newton's methods. The logarithmic barrier is one of the most common barrier functions that is used in interior-point methods, and it has the form:

$$\phi_j(\boldsymbol{\theta}) = -\kappa \log(-\mathcal{C}_j(\boldsymbol{\theta})) \tag{2.26}$$

where κ is a parameter that tunes the approximation. The lower the κ, the better the approximation. Obviously, Eq. (2.26) is only valid when $\mathcal{C}_j(\boldsymbol{\theta}) < 0$. Thus in order for this approach to work correctly, it requires the initial guess to be a point inside the feasible set (this is why this approach is called interior-point). Using the barrier functions, the optimization problems (2.25) can be formulated as:

$$\begin{aligned} &\text{minimize} && J(\boldsymbol{\theta}) + \sum_{j=1}^{n_c} \phi_j(\boldsymbol{\theta}) \\ &\text{subject to} \\ & && \mathcal{L}_i(\boldsymbol{\theta}) = 0 \qquad i = 1..n_l \end{aligned} \tag{2.27}$$

In the optimization problem (2.27), the inequality constraints are implicitly presented in the objective function. Note that the equality constraints cannot be formulated with barrier functions. But this is not problematic since they can efficiently be tackled through using lagrange multipliers.



(a) Convergence to the global minimum **(b)** Convergence to a local minimum

Figure 2.5: An example of using an interior point algorithm to solve a non-convex optimization problem. As it is expected, due to the non-convexity, different solutions are obtained by starting the optimization from different initial points. In this graph, the optimization's initial and final points are illustrated with black diamond and star, respectively. The optimization's step is indicated by circle. The contour lines of the objective function are shown in orange. The infeasible region (i.e. the region where the optimization's constrain does not hold) is shaded in blue. For more details, please refer to Section 2.3.

The optimization problem (2.27) is an approximation of (2.25). However, this approximation gets more and more accurate as $\kappa \to 0$. Irrespective of the value of κ, the optimization constraints are always fulfilled because the logarithmic barrier grows without bound as $\mathcal{C}_j(\boldsymbol{\theta}) \to 0$. Given an initial guess inside the feasible set, and a large κ, a basic interior-point method is a two-step procedure: 1) find an optimal of the problem (2.27) using a Newton method, 2) if κ is less than a small threshold, stop the algorithm. Otherwise, reduce it and go to step one. Recent interior-point methods combine the two steps into one single step and decrease κ at each iteration of the Newton's method (R. A. Waltz et al., 2006). To improve the optimization performance, it is possible to use a line search method to adaptively change the magnitude of movement along the descend direction (Kelley, 1999).

Figure 2.5 shows an example of using a simple interior-point algorithm that follows the above two-step procedure to solve the following problem:

$$\begin{aligned} &\text{minimize} && J(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \begin{bmatrix} 1 & 0.8 \\ 0.8 & 2 \end{bmatrix} \boldsymbol{\theta} \\ &\text{subject to} \\ & && \boldsymbol{\theta}^T \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \boldsymbol{\theta} < 0 \end{aligned}$$

In the above problem, the optimization's constraint is non-convex (this can also be visually verified in Fig. 2.5). As described before, when solving a non-convex problem, not only convergence to the global minimum cannot be ensured, but also the optimization may converge to different local minima by starting from different initial positions (see Fig. 2.5).

2.4 Robot Motion Control

As outlined before, in this thesis we consider robot motions that are defined at the kinematic level. Therefore, in order to be able to execute the desired motion on a real robot, our approach relies on a low level tracking controller to convert kinematic variables into motor commands. In this section, we explain the approach that we take in this thesis to perform this procedure.

Robot control techniques deal with the question of how to generate a sequence of actuator commands (e.g. force or torque) so as to successfully execute a commanded task. Generally for robotic systems, a task is defined in either Joint (also called Configuration) or operational spaces. In case of the former, the problem of robot control is reduced to designing a tracking controller that allows following a path $\boldsymbol{p}(t)$ as close as possible to a reference trajectory $\boldsymbol{p}^r(t)$.

When the task is defined in the operational space, there are two ways to control the robot: 1) Using an *inverse kinematics algorithm* to transform operational space references into joint space references, and then utilizing a joint space controller to generate motor commands so as to follow the transformed path, 2) Using an operational space controller to directly generate the required motor commands from the operational space references.

The choice between these two types of controllers depend on both the platform and the task at hand. Some platforms rely solely on their built-in controller and only accept reference trajectories that are defined in joint space (the so-called position-controlled robots). Hence, for these platforms, the joint space controller is the sole available possibility. More advanced robots allow the user to directly send control commands to the robot (the so-called torque-controlled robots), and thus the user has the option to choose the most proper control scheme based on the task at hand.

Regarding the task, the use of a joint space control scheme usually suffices for motion control in the free space, whereas the choice of operational space control is more pressing when we deal with the problem of controlling interaction between the robot and the world (Siciliano et al., 2009). In this thesis we take the former as the main focus of this thesis is on the problem of motion generation in the free space (i.e. without any physical interaction with the world). Furthermore, three out of the five robots that we use in this thesis are only position-controlled (The Hoap-3, iCub, and Katana robots), and thus the choice of joint space controller is inevitable. As the use of an operational space controller is insignificant in our implementation and for consistency between our platforms, we choose a joint space control scheme to control all the five robots considered in this thesis.

Next, we review the pseudo-inverse kinematics algorithm that we use to transform operational space references to joint space references. Then we describe the two control approaches that are adopted in this thesis, namely PID

and inverse dynamics controls, to generate the required torque commands to execute the desired motion (defined in joint space).

2.4.1 Inverse Kinematics

Consider a robotic arm with n_q Degrees of Freedom (DoF) that is required to follow a reference trajectory in a d-dimensional operational space. For any robotic arm, the position and orientation of the end-effector $\boldsymbol{p} \in \mathbb{R}^d$ can directly be determined in terms the arm configuration $\boldsymbol{q}$ through the kinematic function $\boldsymbol{k}$:[5]

$$\boldsymbol{p} = \boldsymbol{k}(\boldsymbol{q}) \tag{2.28}$$

Inverse kinematics algorithms deal with the opposite problem, i.e. it consists of determining a set of joint angles $\boldsymbol{q}$ that corresponds to a particular end-effector position $\boldsymbol{p}$. Contrary to Eq. (2.28) that can be computed in a unique manner, the inverse kinematics problem is non-trivial generally due to the absence of a closed form solutions. Furthermore, the existence of several or infinite solutions in some situations makes the above problem even more difficult.

Differential kinematics is one possible way to solve the inverse kinematics problem (Siciliano et al., 2009). This technique exploits the relationship between joint velocities and the corresponding end-effector velocity to solve the above problem. This relationship is conveyed through the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$ which can directly be derived by differentiating from Eq. (2.28):

$$\begin{aligned} & \frac{\partial}{\partial t}\big(\boldsymbol{p} = \boldsymbol{k}(\boldsymbol{q})\big) \\ \Rightarrow \quad & \dot{\boldsymbol{p}} = \frac{\partial \boldsymbol{k}(\boldsymbol{q})}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} \\ \Rightarrow \quad & \dot{\boldsymbol{p}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned} \tag{2.29}$$

where $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{d \times n_q}$. For redundant manipulator (i.e. when $d < n_q$), there exists infinite solutions to Eq. (2.29). In this situation, often the problem is formulated as a constrained linear optimization problem to determine an optimal solution that requires the least movement in joint space:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ \text{subject to} \quad & \dot{\boldsymbol{p}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \end{aligned} \tag{2.30}$$

The optimal solution to the above problem is given by:

$$\boldsymbol{q}^* = \boldsymbol{J}^\dagger \boldsymbol{p} \tag{2.31}$$

[5] Note that depending on the task at hand, the vector $\boldsymbol{p}$ may contain both position and orientation of the end-effector, or solely either of them. In this section, for simplicity, we consider $\boldsymbol{p}$ only contains the positional information. However, all the following formulations can be applied when it also has the orientational part.

where

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{T}\left(\boldsymbol{J}\boldsymbol{J}^{T}\right)^{(-1)} \tag{2.32}$$

is the right pseudo-inverse of $\boldsymbol{J}$. Note that for the clarity of the formulations we have removed the dependency on $\boldsymbol{q}$ in the above equations.

Given a reference trajectory $\boldsymbol{p}^{d}(t)$ in operational space with $\boldsymbol{p}^{d}(0) = \boldsymbol{k}(\boldsymbol{q}(0))$, Eq. (2.31) can iteratively be used to determine the proper sequences of robot joint angles to execute the task. There are several extensions to Eq. (2.31) which we briefly describe the two popular ones here. For more details please refer to (Sciavicco & Siciliano, 2000).

2.4.1.1 Singularity Problem:

The singularity problem can be overcame by using the so-called *damped least-squares inverse*:

$$\boldsymbol{J}^{\ddagger} = \boldsymbol{J}^{T}\left(\boldsymbol{J}\boldsymbol{J}^{T} + \lambda^{2}\boldsymbol{I}\right)^{(-1)} \tag{2.33}$$

where $\boldsymbol{I} \in \mathbb{R}^{n_q \times n_q}$ is the identity matrix, and λ is a damping factor controlling the upper-bound on velocity.

2.4.1.2 Satisfying Additional Constraints:

When working with redundant robots, it might also be possible to satisfy other constraints in addition to Eq. (2.29). This can be obtained by projecting the additional constraint(s) into the null space of $\boldsymbol{J}$. Consider the vector $\dot{\boldsymbol{q}}^{c}$ corresponding to an extra constraint (e.g. increasing the manipulability measure, the distance from mechanical joint limits, or the distance from an obstacle), then the inverse kinematics problem leads to the following solution:

$$\boldsymbol{q}^{*} = \boldsymbol{J}^{\dagger}\boldsymbol{p} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}^{c} \tag{2.34}$$

2.4.2 PID Control

There are several ways of building a tracking controller including different variants of Proportional-Integral-Derivative (PID) control (Ogata, 2001), inverse dynamics control (Siciliano et al., 2009), adaptive and robust controls (Ioannou & Sun, 1996), optimal control (Bryson & Ho, 1975), etc . Among these methods, the PID controller is one of the most common control approaches that generates a control command $\tau(t)$ based on the error between the reference and actual trajectories, i.e. $e(t) = \boldsymbol{p}^{r}(t) - \boldsymbol{p}(t)$. A PID controller has the following form:

$$\tau(t) = K_p e(t) + K_d \frac{d}{dt} e(t) + K_i \int_0^t e(\tau) d\tau \tag{2.35}$$

where K_p, K_i, and K_d are positive gains to adjust the control behavior. The three terms in Eq. (2.35) respectively compensate for the present, the accumulated sum of the past, and the prediction of the future errors. PID controllers can be used without knowledge of a robot's dynamics. However in this case, the controller may exhibit a poor tracking performance, and the the system's stability may no longer be ensured. In the iCub, Hoap 3, and Katana robots that are used in this thesis, the motion control is purely based on a PID controller (because low-level access to motor commands is not provided). For these robots, the PID gains are tuned so as to allow executing a wide range of motions provided the reference trajectory is smooth.

2.4.3 Inverse Dynamics Control

When the knowledge about the robot dynamics is available, one can use more advanced control schemes to leverage the system's performance by exploiting this information. Inverse dynamics control is one of the nonlinear model-based control techniques that can considerably improve the trajectory tracking performance. This approach is founded on the idea of obtaining an exact linearization of system dynamics by means of a nonlinear state feedback (Siciliano et al., 2009). In the absence of external end-effector forces and the static friction, the equations of motion of a manipulator can be described by:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{2.36}$$

where $\boldsymbol{q}$ is the vector of robot's joints angle, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ correspond to its first and second time derivatives, $\boldsymbol{M}(\boldsymbol{q})$ accounts for inertial terms, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ represents centrifugal and Coriolis effects, $\boldsymbol{F}_v$ stands for viscous friction coefficients, $\boldsymbol{g}(\boldsymbol{q})$ is the gravity compensation terms, and $\boldsymbol{\tau}$ is the vector of robot's torque commands. Eq. (2.36) can be rewritten as:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \tag{2.37}$$

where

$$\boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) \tag{2.38}$$

Consider an auxiliary DS with a new input vector $\boldsymbol{u}$ as follows:

$$\ddot{\boldsymbol{q}} = \boldsymbol{u} \tag{2.39}$$

The system under Eq. (2.39) is both linear and decoupled with respect to the new input $\boldsymbol{u}$. This system is called stabilizing linear control, and as it appears from its name, it stabilizes the overall system. Given position $\boldsymbol{q}^r$, velocity $\dot{\boldsymbol{q}}^r$, and acceleration $\ddot{\boldsymbol{q}}^r$ of the reference trajectory, the input $\boldsymbol{u}$ is defined according to:

$$\boldsymbol{u} = \ddot{\boldsymbol{q}}^r + \boldsymbol{K}_p(\boldsymbol{q}^r - \boldsymbol{q}) + \boldsymbol{K}_d(\dot{\boldsymbol{q}}^r - \dot{\boldsymbol{q}}) \tag{2.40}$$

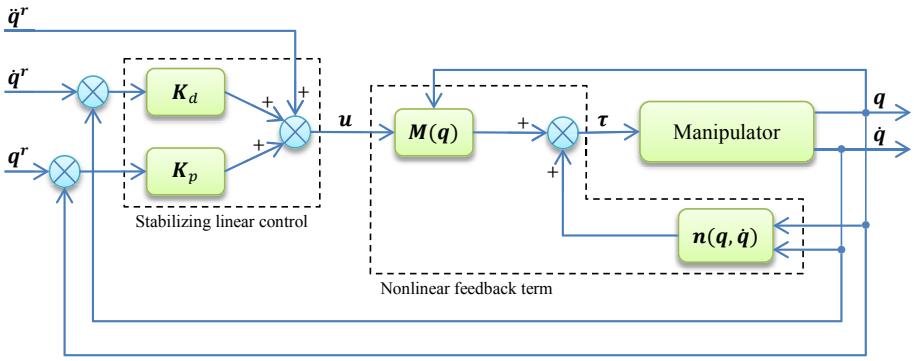


Figure 2.6: Block diagram of an inverse dynamics controller.

where $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are positive definite matrices. Using Eqs. (2.37), (2.39) and (2.40), the manipulator control $\boldsymbol{\tau}$ can be then described in terms of manipulator state, the tracking error, and the acceleration of the reference trajectory:

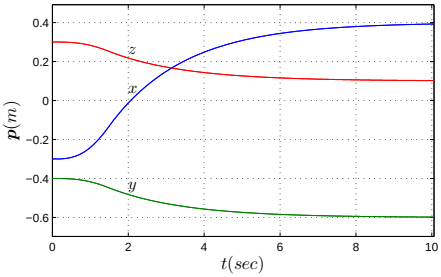
$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{u} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{2.41}$$

Eq. (2.41) corresponds to a nonlinear control law that is termed *inverse dynamics control*. Fig. 2.6 shows the block diagram of this controller. As can be observed, this controller encompasses two feedback loops: an inner loop that provides a nonlinear feedback term based on the robot's dynamics, and an outer loop that operates on the tracking error.

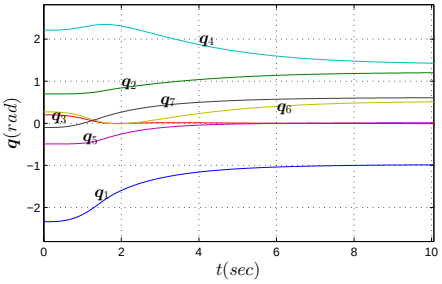
In general, inverse dynamics control exhibits a better performance than a pure PID controller, provided an accurate model of the robot is available. In this thesis, the model of the two robot arms (i.e. the WAM and DLR arms) that are controlled with inverse dynamics control is accurate enough to execute the considered tasks. Nevertheless, in case of imperfect modeling, one can adopt more advanced control approaches such as robust or adaptive controls to compensate for such inaccuracies (Siciliano et al., 2009).

Figure 2.7 shows an example of using an inverse dynamics controller on 7-DoF Barrett WAM arm (see Fig. 2.7a). In this example, the robot should follow a desired trajectory in the task space while preserving the end-effector's orientation throughout the motion. The joint angles are computed using the damped least squares pseudo-inverse kinematics as described in Section 2.4.1. The joint torques are determined using the presented inverse dynamics controller. As can be seen in Fig. 2.7b, the controller shows a very good performance as the difference between the reference and the robot trajectories is insignificant (the maximum tracking error is $5.73 \times 10^{-4} m$).

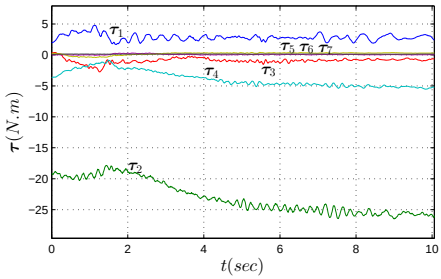
(a) The 7-DoF Barrett WAM arm



(b) Position in the operational space



(c) Joint angles



(d) Joint Torques

Figure 2.7: An example illustrating the execution of a path $\boldsymbol{p}(t)$, that is defined in the operational space, on the 7-DoF Barrett WAM arm. The joint angles and the torque commands are computed using the damped least squares pseudo-inverse kinematics and an inverse dynamics controller, respectively. The maximum tracking error is $5.73 \times 10^{-4} m$.

Chapter 3

Background Research

In theory, there is no difference between theory and practice. But in practice, there is.

Yogi Berra

The problem of motion generation for robot movement has been an active research topic in robotics for years, and many techniques have been suggested addressing different aspects of this problem. These approaches have been a valuable source of inspiration for this thesis, and assessing their perspectives in tackling the problem mentioned above and their pros and cons are helpful to delineate the contribution of this work. This section is targeted at reviewing these techniques with an emphasis on the work particularly devised to generate discrete movements. This overview is not exhaustive and is not aimed to provide a complete account of what has been done within this domain. Instead, it is intended to provide the reader with enough information to situate this work among the relevant state of the art approaches. For interested readers, wherever it is possible, references to more complete reviews are provided.

For clarity of this chapter, we focus our review in each section on techniques that share some common ground. Nevertheless, due to multi-functionality of some of these approaches, they are appeared in two or more sections. This chapter unfolds as follows: In Section 3.1 we give a brief account of global planning approaches for motion generations. These approaches are well-known for their ability to find a feasible path (if it exists) in static complex environments. In Section 3.2, we describe the techniques that aim at unifying planning and execution into a single strategy, which especially suit them for use in dynamic environments. In Section 3.3, we present the methods for robot motion generations that are grounded on imitation learning. The works that are addressed in this section are those that are closest to the framework we present in Chapters 4 and 5. Finally in Section 3.4, we review the techniques that can be used for obstacle avoidance.

3.1 Planning Approaches

Path planning approaches deal with the problem of finding a collision-free path from an initial state $\boldsymbol{\xi}^0$ to a final state $\boldsymbol{\xi}^*$ given a complete description of a robot's geometry and its environment. Basically, path planning is a pure geometric problem that is solved in the robot Configuration-space ($\mathcal{C}$-space). $\mathcal{C}$-space modeling is advantageous in that it offers an abstract way of solving the planning problem by mapping a complex shape robot into a single point (Udupa, 1977).

While solving a complete geometric-based planning problem is possible (e.g. see (Lozano-Perez & Wesley, 1979; Schwartz & Sharir, 1983; Canny, 1988)), they are computationally very expensive and thus unsuitable for practical applications. Quite often, determining an exact geometric modeling of the $\mathcal{C}$-space is non-trivial. The difficulty is mainly due to the high dimensionality of the $\mathcal{C}$-space and the absence of an easy and direct way to describe the robot workspace in this space (Kavraki & LaValle, 2007). Sampling-based planning techniques are suggested to partly overcome these difficulties and to make the computational complexities of finding a feasible path more tractable, but at the cost of providing a lower level of completeness in the sense that they cannot detect if no path can be found.

The Probabilistic Roadmap Method (PRM) (Kavraki et al., 1996; Boor et al., 1999; Lien et al., 2003; S. LaValle et al., 2004; D. Hsu et al., 2006) and Rapidly exploring Dense Trees (RDT) (Kuffner & LaValle, 2000; Strandbergtrees, 2004; Yershova et al., 2005; Zucker et al., 2007) are the two best known examples of sample-based planning methods. The former begins by constructing a roadmap that describes the connectivity properties of the free region in $\mathcal{C}$-space. After its construction, it then uses the roadmap to answer multiple queries. In contrast, RDT-based approaches attempt to incrementally build the tree data structure online by exploring the part of the $\mathcal{C}$-space that will lead to solving a single query point as fast as possible. Both PRM and RRT are probabilistically complete in the sense that the probability that the planner fails to return a solution, if one exists, decays to zero as the number of samples approaches infinity (Karaman & Frazzoli, 2011).

As outlined before, conventional planning approaches only solve a pure geometric path planning problem. However, there are other constraints such as the velocity limit that the robot needs to conform to during the task execution. When a planning problem considers constraints on, at least, velocity and acceleration, it is often referred to as kinodynamic planning (Donald et al., 1993). Solving a kinodynamic problem is much harder than a pure geometric problem since it often requires several discretizations (e.g. discretization of the control input, the time interval, etc.) and usually acts in higher dimension. Using sampling-based planners (D. Hsu et al., 2002), utilizing so-called motion

primitives (Go et al., 2004), and decoupling the kinodynamic problem into a set of subproblems (Ferbach, 1996) are some common attempts to reduce the computational complexity of kinodynamic planners.

Despite the power of global path planning algorithms in ensuring to find a valid solution (if it exists), they virtually operate in open-loop as they only provide a pre-planned trajectory based on the current description of the environment (S. M. LaValle, 2006). Thus, by using these techniques a task implementation is in fact split into two phases: the planning and the execution. If any change happens in the environment during the second phase (e.g. the goal state is changed, an obstacle appears, etc.), a re-planning step is thus required. Due to the computational complexity of these approaches, the re-planning cannot usually be performed in realtime and thus makes these approaches practically unsuitable for implementation in dynamic environments.

3.2 Feedback Motion Planning Approaches

In many real world experiments it is crucial to have some form of feedback in order to handle modeling errors as well as to adapt to unpredictable future events during execution of a task. The successfulness of this idea has been proven in control theory across numerous applications. Given this widespread success, it seams valuable to utilize this control scheme in the context of motion planning (S. M. LaValle, 2006).

Feedback motion planning approaches were suggested to unify planning and execution into a single motion strategy. As a result, these approaches provide the required reactivity to adapt to dynamic environments. Traditionally, there are two main differences between feedback motion planning approaches and the techniques developed in control theory: 1) Motion planning approaches do not consider the dynamics of the robot during the planning, and 2) Control theory is preliminary concerned with the problems such as stability, optimality of the control command, giving less emphasis on issues such as obstacle avoidance[1].

By ignoring the dynamics of the robot when generating kinematic references, feedback motion planning approaches implicitly rely on the assumption that the differential constraints can be appropriately handled through refinements during the execution of the task[2]. This simplification allows focusing on the planning problem which has yielded several rigorous techniques that can ensure convergence to the target in cluttered environments.

[1] It should be noted both fields are expanding their scope, and thus the above differences are fading away. See (S. M. LaValle, 2006) for more discussion on similarities and differences between control and planning approaches.

[2] For example, consider the robot is at point $\boldsymbol{\xi}^A$ and is required to move at speed $\dot{\boldsymbol{\xi}}^A$ to reach the point $\boldsymbol{\xi}^B$. However, due to hardware limitations, the robot cannot move at the commanded speed and thus ends up in a different position $\boldsymbol{\xi}^C$. Thanks to the feedback term, the planner adopt a new trajectory from this point and guides the robot to the goal point.

The Potential field approach is one of the earliest works on feedback motion planning (Khatib, 1986). In this approach, the robot is considered as a particle, and its workspace is described by a global potential function whose gradient leads the robot to the target point. More specifically, the global potential function is defined as a sum of an attractive potential function located at the target, and a set of repulsive potential functions representing obstacles. The direction of the movement (i.e. the gradient direction) is thus governed by the net force induced due to the presence of all these fields. Potential functions are subject to local minima, i.e. they cannot ensure the target is always reachable.

A potential function that is free from local minima is called navigation function (Rimon & Koditschek, 1992). For continuous problems, such as reaching motions, it is often very difficult to find a single navigation function to describe the task at hand, and the earliest approach were only applicable to simple environments (Koditschek, 1987; Rimon & Koditschek, 1992). One possible way to overcome this difficulty is to transfer the continuous problem into its discrete counterpart. This simplification allows to build an estimate of navigation functions by performing a backward search from the goal point using different discrete planning techniques such as Dijkstra's algorithm (Dijkstra, 1959), A-star (Hart et al., 1968), etc. The obtained navigation function is called an approximate or a grid-based navigation function. Due to the computational complexities, approximate navigation functions are often used in mobile robots, which usually work on lower dimensions.

Instead of building an approximate navigation function, more recent approaches suggest using a piecewise-smooth navigation function that is defined over a collection of simple-shaped cells (Conner et al., 2003; Lindemann & LaValle, 2005), for instance see Fig. 3.1. These approaches, in essence, take a hierarchical form by decomposing the problem into two subproblems: 1) Defining a discrete planning problem over all the cells, which will be used to provide high-level information on how to reach the final cell that contains the goal, and 2) Considering a simple navigation function over each cell that derives all the motion within the cell to the next (neighbor) cell that is determined from the first step. These approaches can also be viewed as hybrid systems since their composition requires planning to ensure global convergence, yet controlling within each cell can be done simultaneously without any need to re-plan. An example of such a navigation function is illustrated in Fig. 3.1. It should be noted that as transitions over cells impose discontinuity, special consideration should be taken to reduce its effect as much as possible (Lindemann & LaValle, 2005). Although piecewise-smooth navigation functions provide us with an exact (as opposed to approximate) solution, their application is still limited to lower dimensional problems or to tasks that have some special structure (S. M. LaValle, 2006). Furthermore, as each navigation function corresponds to a fixed situation, any change in the environment (e.g. moving obstacles) requires regeneration of a new navigation function.

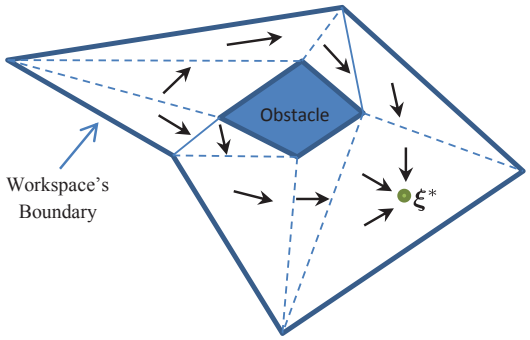
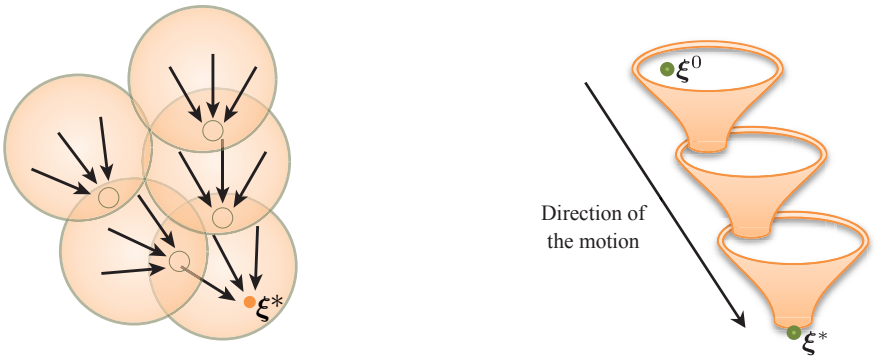


Figure 3.1: An example illustrating a piecewise-smooth navigation function that is defined over a collection of simple-shaped cells. The dashed lines shows the border of each cell, and the direction of motion within each cell is indicated by an arrow. The composition of cells is performed by a discrete planner to ensure convergence of all trajectories to the target $\boldsymbol{\xi}^*$. Note that in this specific example, triangular shaped cells are chosen, but different shapes can be taken depending on the task at hand.

The funnel-based approaches are other techniques that are suggested to construct a continuous navigation function by decomposing it into a sequence of overlapping funnels (Choi & Latombe, 1991; Conner et al., 2006; L. Yang & LaValle, 2004). The motion in each funnel is derived by a potential function, which is designed so as to ensure convergence to the next funnel. To quickly determine the funnel that contains the current state $\boldsymbol{\xi}$ (which is essential during the execution), simple shapes (e.g. a sphere in 3D case) are used to describe the region of attraction of each funnel. Sampling techniques are often used to generate funnels so that they cover the free space of the robot as much as possible. Similarly to the approaches described above, discrete planners can be used to compose funnels so that each funnel guides the robot to the next one till the robot reaches the target (see Fig. 3.2). In essence, the two approaches are very similar. The only difference is that the former allows neighbor regions to overlap as opposed to the latter that partitions the free space into distinct regions (that only share a common border). The application of funnel-based techniques to higher dimensions and dynamic environments is still a work under progress.

Harmonic Potential functions (Connolly et al., 1990; J.-O. Kim & Khosla, 1992; Feder & Slotine, 1997) are another family of navigation functions that their formulation is inspired by the description of the dynamics of some physical processes such as heat transfer or fluid flow. Despite ensuring the global convergence to the target, construction of an exact navigation function is limited to simple environments with obstacles of specific shapes. Approximate methods based on discretized space overcome this limitation but at the cost of being computationally more expensive (Brock et al., 2007).

Movement primitive approaches are also another type of feedback planning techniques that can be used to control robot motions in dynamic environments (Ijspeert et al., 2002b; Dixon & Khosla, 2004b; Billard et al., 2008). Each movement primitive codes a behavior (such as reaching for a cup, swinging a golf club, etc.) with a set of autonomous or non-autonomous differential equations. These



(a) Compositions of funnels. **(b)** Evolution of the motion in time.

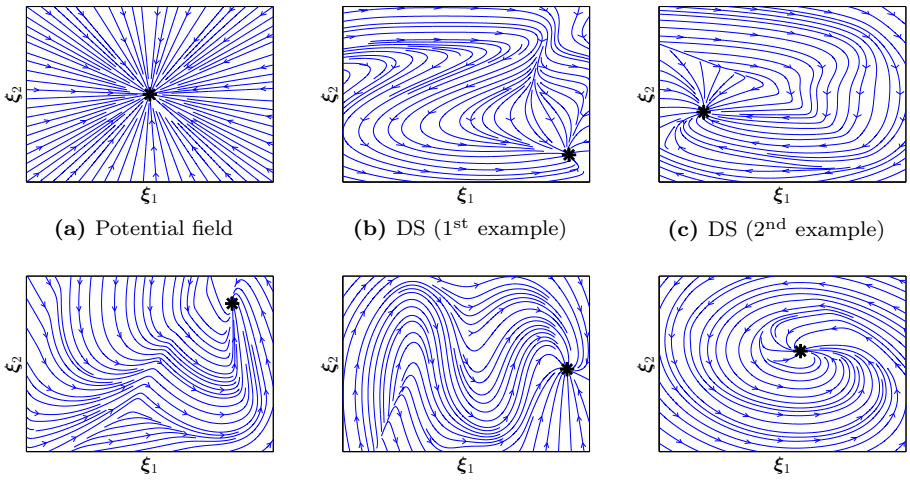
Figure 3.2: A navigation function that is defined as a composition of funnels. The small circles indicate the exit end of the funnel. Starting from an initial point $\boldsymbol{\xi}^0$, the motion passes from one funnel to the next one till the robot reaches the target $\boldsymbol{\xi}^*$.

techniques are often referred to as Dynamical System-based approaches since they directly define a robot motion with a differential equation (as opposed to the potential field approaches that first define an energy function and then take its gradient to generate the motion). We also use this terminology throughout this thesis; however, it should be noted that all the techniques presented above are in fact a DS approach.

As outlined in Section 2.1, when defining a motion with DS, ensuring its global or local asymptotic stability at the target is crucial in order to provide a useful control policy. One possible way to verify this is to find a Lyapunov energy function for the DS at hand (see Theorem 2.2). A Lyapunov function, in essence, is a navigation function in the absence of obstacles. Despite this similarity, in contrast to the potential field methods, DS-based approaches offer a means to generate customized motions (i.e. control the way trajectories approach the target). Figure 3.3 highlights the difference between these approaches through a simple reaching example in the absence of obstacles. Here, we consider a quadratic Lyapunov (navigation) function that is defined by:

$$V(\boldsymbol{\xi}) = (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} (\boldsymbol{\xi} - \boldsymbol{\xi}^*) \tag{3.1}$$

where $\boldsymbol{\xi}^*$ is the target point. By taking the gradient of $V(\boldsymbol{\xi})$, the potential field approach generates motions that move on a straight line towards the target (see Fig. 3.3a). However, by taking a DS-based approach, there are infinite ways to approach the target. For example, Figs. 3.3b to 3.3f illustrate five different ways of customizing the motion to the target. Stability of all these motions can be ensured using the Lyapunov function that is given by Eq. (3.1). Although the solution from the potential field may seem adequate for mobile robots, it is far too limiting to model human-like discrete robot motions. Since the main focus of this thesis is to devise a DS-based framework for manipulators, we will discuss the DS-based approaches in more detail later on in Section 3.3.2.



(a) Potential field **(b)** DS (1^{st} example) **(c)** DS (2^{nd} example)

(d) DS (3^{rd} example) **(e)** DS (4^{th} example) **(f)** DS (5^{th} example)

Figure 3.3: Comparison between the potential field and DS approaches. In **(a)**, we use the potential function given by Eq. (3.1) to generate motions. This function is also used to ensure stability of DS motions that are given in **(b)-(f)**. In this example, the target point is shown with a black star. As can be seen, while the potential field approach solely provides one way to reach the target, the DS approach can be used to form the basin of attraction at the target.

3.3 Imitation Learning Approaches

In imitation learning, robots are taught to perform a task by observing a set of demonstrations provided by a teacher (human or robot). Demonstrations to a robot may be performed in different ways: back-driving the robot, teleoperating it using motion sensors, or capturing a task via vision sensors. The learning process consists of extracting the relevant information from the demonstrations and encoding this information into a motion model that can be used to reproduce the task. Imitation learning has been used for various applications including: software development (Cypher, 1993; Lieberman, 2001; Mitchell et al., 1994), symbolic learning and reasoning (Lozano-Perez, 1983; Hovland et al., 1996; Pardowitz et al., 2007), and motion modeling (Ude, 1993; Andersson, 1989; Calinon et al., 2007; Kulic et al., 2008). As the main focus of this thesis is on the latter, we will next review works along the topic of motion modeling. In our review, we consider two general directions of work, namely time-indexed and DS-based modelings. The former includes major body of work in imitation learning, while the latter has been recently introduced to provide a robust means of encoding robot motions.

3.3.1 Time-indexed Trajectory Modeling

Traditional means of encoding trajectories is based on spline decomposition after averaging across training trajectories (Hwang et al., 2003; Andersson, 1989;

Aleotti & Caselli, 2006; Ude, 1993). While this approach provides a useful tool for quick and efficient decomposition and generalization over a given set of trajectories, it is however heavily dependent on heuristics for segmenting and aligning the trajectories and gives a poor estimate of nonlinear trajectories.

Some alternatives to spline-based techniques perform regression over a non-linear estimate of the motion using different regression techniques (Delson & West, 1996; Ogawara et al., 2003; Muehlig et al., 2009; Yamane et al., 2004; Calinon et al., 2007; Schaal & Atkeson, 1994). A number of other approaches are also founded on Hidden Markov Models (HMMs) to encode temporal and spatial variations of robot motions (Tso & Liu, 1996; Inamura et al., 2002; J. Yang et al., 1997; Kulic et al., 2008; Calinon & Billard, 2005). Further works on trajectory modeling address the problem of extracting a task's constraint from multiple demonstrations (Calinon & Billard, 2007b,a), or to learn a local model of the robot's dynamics along with inferring the desired trajectory (Coates et al., 2008).

Although the methods described above provide powerful means for encoding multi-dimensional nonlinear trajectories, similar to spline-encoding, they depend on explicit time-indexing and virtually operate in an "open-loop". Time dependency makes these techniques very sensitive to both temporal and spatial perturbations. To compensate for this shortcoming[3], one requires evaluating a heuristic to re-index the new trajectory in time, while simultaneously optimizing a measure of how good the new trajectory follows the desired one. Finding a good heuristic is highly task-dependent and a non trivial task, and becomes particularly non intuitive in high-dimensional state spaces.

3.3.2 Dynamical System-based Modeling

An important concept in imitation learning is the ability to generalize the task and to adapt it to a new situation. This concerns the problem of performing the task under different circumstances than those present during demonstrations, which is desirable mainly for two reasons: 1) The number of demonstrations can be kept small, and 2) Given appropriate adaptation, an acquired skill can be used to carry out a more complex task than the teacher is capable of demonstrating.

Dynamical system-based approaches to modeling robot motions have been recently advocated as a powerful alternative to trajectory-based techniques as they offer a powerful tool for robust control of robot motions from a small set of demonstrations. They ensure high precision in reaching a desired target, yet can be easily modulated to generate new motions in areas not seen before. Moreover, they provide an inherent robustness to perturbations and instant adaptation to changes in the environment (Billard et al., 2008).

As outlined before in Section 3.2, the DS approach to modeling robot motions is a type of feedback motion planning. Hence a controller driven by a DS is

[3] If one is to model only time-dependent motions – i.e. motions that ought to be performed in a fixed amount of time – one may prefer using a time-dependent encoding.

robust to perturbations because it embeds all possible solutions to reach a target into one single function. Such a function represents a global navigation map which specifies on-the-fly the correct direction for reaching the target, considering the current position of the robot and the target. During the last decade, DS has been used to model discrete motions (Ijspeert et al., 2002b; Calinon, D'halluin, et al., 2010; Kulic et al., 2008; Schaal et al., 2004; Pastor et al., 2009; Ude et al., 2010), rhythmic motions (Schaal et al., 2004; Ijspeert et al., 2002b; Righetti et al., 2006; Rochat et al., 2011), hitting motions (Calinon, Sauser, et al., 2010; Kober, Mulling, et al., 2010), etc.

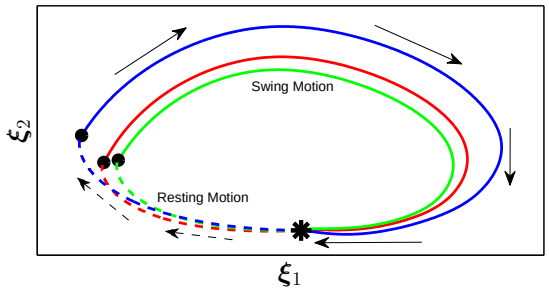
In DS approaches, the control policy to drive a robotic platform is modeled with a first or higher order DS. When controlled through a DS, a robot motion unfolds in time with no need to re-plan. An estimate of the DS can be built from a few demonstrations of the task at hand. The estimated DS captures the invariant features in the user demonstrations, and can generate motions that resemble the user demonstrations (Billard et al., 2008).

Each DS model codes a specific motion (behavior), and is called a movement primitive (also known as motor primitive). They can be seen as a building block that can be used to generate more complex or new motions through sequencing or superimposition of the primitives. This modularity of DS-based movement primitives is essential as it allows controlling a wide repertoire of movements from a (small) set of basic motions (Schaal, Ijspeert, & Billard, 2003; Wolpert & Kawato, 1998). Figure 3.4 shows two examples of exploiting this modularity of movement primitives to generate new motions[4].

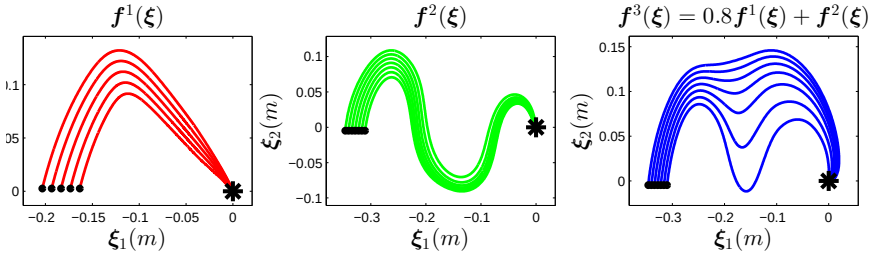
The idea of using a programmable DS formulation to generate motions is not new and has been an active topic for decades. The Vector Integration To Endpoint (VITE) model is one of the early approaches that is suggested to simulate arm reaching movements (Bullock & Grossberg, 1988b,a; Gaudiano & Grossberg, 1992; Bullock et al., 1999). This model has a simple structure with two control parameters: a 'target length' and a 'go command'. The former specifies the desired length of the muscle, while the latter controls the onset of the motion and its speed profile. One of the features of the VITE model is that irrespective of the target length, all muscles reach their respective desired length at the same time, which is interesting for synchronized movements.

Central Pattern Generators (CPGs) are also another type of DS model that are suggested to model rhythmic behaviors (Grillner, 1985; Raibert, 1986; Delcomyn, 1980; Marder & Bucher, 2001; Ijspeert et al., 1998). Being inspired from the neurobiology of invertebrate and vertebrate animals, CPGs are able to generate basic rhythmic movements without any need to receive any rhythmic input. They also accept higher-level stimuli which can be used to modify their rhythmic behavior to adapt to different situations (e.g. to increase the speed of movements or to generate different locomotion gates). Several CPG models

[4]Remark that nonlinear sum of two or more stable DS is not necessary stable, and especial attention should be considered in this regard (see Section 4.2 for further discussion).



(a) This graph shows an example of a tennis swing which is composed of a swing and a resting phase. The motion in each phase is encoded as a basic movement primitive, and the complete tennis swing motion is thus obtained by sequencing these two primitives. The trajectories generated by the swing and resting models are shown in solid and dashed lines, respectively. The direction of the motion is indicated by arrows. Only three examples of generated trajectories are shown here (plotted as red, green, and blue lines).



(b) In this example $f^1(\xi)$ and $f^2(\xi)$ are two basic movement primitives that represent an angle and sine-shaped motion, respectively. The new movement primitive $f^3(\xi)$ that includes a mixture of both behaviors is obtained through a linear superposition of $f^1(\xi)$ and $f^2(\xi)$.

Figure 3.4: Illustration of two examples exploiting modularity of DS models to generate **(a)** a more complex motion and **(b)** a new movement primitive. In this figure, the black star and circles indicate the target and initial points, respectively.

have been developed so far to model robots locomotion. Study of these models is beyond the scope of this thesis, and we refer interested readers to (Ijspeert, 2008) for a survey on the CPG models.

Recurrent Neural Network (RNN) and its variants (Lukosevicius & Jaeger, 2009; Reinhart & Steil, 2011; Lin et al., 1995; Sudareshan & Condarcure, 1998; B. Pearlmutter, 1989; Ito & Tani, 2004) are another fascinating DS-based tools that have been used to model discrete and rhythmic motions (besides to its application in other domains such as signal processing, vision systems, system identification, etc.). In general terms, RNNs are computational models that are composed of numerous interconnected neurons. As it appears from its name, connection topology in RNN includes cycles which allows to render it to be a DS. There are several known variants of RNN such as Hopfield networks (Hopfield, 1982, 2007), Boltzmann machines (Hinton, 2007; Ackley et al., 1985), Reservoir Computing (Lukosevicius & Jaeger, 2009; Maass et al., 2002), etc. Consequently, different training algorithms have also suggested for RNNs. An overview of different RNN structures and training algorithms are presented in (B. A. Pearlmutter, 1995; Atiya et al., 2000; Medsker & Jain, 1999). Despite the potential and capability of RNNs, there are a number of shortcomings associated

to them such as long training times, difficulty in ensuring global asymptotic stability at the target (e.g. in case of reaching motion), bifurcation during learning, and complexities of existing advanced training algorithms (Hopfield, 2007; Atiya et al., 2000; Lukosevicius & Jaeger, 2009).

In the context of robot imitation learning, Schaal et al. (2000) were among the first groups to suggest the idea of using a programmable DS formulation that can be adjusted to different tasks. This idea was then further extended by Ijspeert et al. (2001), where they propose a method, called Dynamic Movement Primitives (DMP), to build an estimate of nonlinear DS via IL. DMP offers a method by which a nonlinear DS can be estimated while ensuring global stability at the attractor. The DS defined by DMP is composed of two main terms: a nonlinear term to accurately encode a given demonstration, and a linear term that acts as a PD controller. These two terms are coupled through a so-called phase variable. Global stability is ensured through exploiting the linear term that takes precedence over the nonlinear part to ensure stability at the end of the motion. The switch from nonlinear to linear dynamics proceeds smoothly according to a phase variable that acts as an implicit clock.

The nonlinear term in DMP is usually learned from a single demonstration using Locally Weighted Regression (LWR) (Atkeson, 1990). DMP offers a robust and precise means of encoding complex dynamics. Its learning phase is fast (a single-shot), and it provides generalization within the region close to the demonstration. These interesting properties have made DMP a popular technique to encode robot motions. It has been originally used to model a forehand swing in tennis (Ijspeert et al., 2002b). Later approaches highlight the use of DMP for different robotics applications such as: walking (Nakanishi et al., 2004; Schaal, Peters, et al., 2003), drumming (Ude et al., 2010; Schaal, 2003), pouring (Pastor et al., 2009; Nemec et al., 2009), flight control (Perk & Slotine, 2006), obstacle avoidance (Park et al., 2008; Hoffmann et al., 2009), lifting (Bitzer & Vijayakumar, 2009), playing table-tennis (Kober, Mulling, et al., 2010), handwriting generation (Kulvicius et al., 2012), etc.

Despite improvements over the past years, the DMP formulation, however, has three general drawbacks: (1) The coupling through phase variable makes the system time dependent and hence sensitive to temporal perturbations. For example, if a perturbation causes some delay in the execution time, this results in having a considerable error in the estimation. These undesirable responses of the system would be avoided if one is able to find a way to reset the phase variable. It is however not easy to determine a robust heuristic for inferring the optimal phase if the motion duration is unknown, e.g. after perturbations. Moreover, the use of heuristic may endanger the asymptotic stability of the system. (2) DMP builds an estimate of DS from a single demonstration. Though this property allows a fast learning algorithm, it can considerably limit the generalization ability of the system to a region close to the demonstration. Thus, when initialized in a point far from the demonstration, or if sustaining large perturba-

tions, the system may no longer generate motions similar to the demonstration. (3) Modeling multi-dimensional systems with DMP is done by learning one DS for each dimension separately, hence neglecting the combined effect of all the dimensions in the motion. As a result, a heuristic is required to synchronize the DS controlling for each dimension, especially when one of the dimensions (e.g. one joint) is perturbed but not the others.

In parallel to DMP, a number of other approaches have been suggested using different types of DS formulations. For instance, Hersch et al. (2008) suggest a hybrid controller composed of two DS working concurrently in end-effector and joint angle spaces, resulting in a controller that has no singularities. While this approach is able to adapt on-line to sudden displacements of the target or unexpected movement of the arm during the motion, the model remains time dependent because, similarly to DMP, it relies on a stable linear DS with a fixed internal clock. An alternative DS approach based on Hidden Semi-Markov Model (HSMM) and Gaussian Mixture Regression (GMR) is also suggested in (Calinon, D'halluin, et al., 2010; Calinon et al., 2011). The method presented there is less sensitive to temporal perturbation thanks to the GMR-HSMM formulation. Asymptotic stability could however not be ensured. Only a brief verification to avoid large instabilities was done by evaluating the eigenvalues of each linear DS and ensuring they all have negative real parts. As stated in (Calinon, D'halluin, et al., 2010) and as we will show in Chapter 4, asking that all eigenvalues be negative is not a sufficient condition to ensure stability of the complete system.

3.4 Obstacle Avoidance

So far we have described different approaches that tackle the problem of modeling discrete movements with an emphasize on their ability to generate trajectories in the absence of obstacles. However, many real world tasks require robotic systems that should work in cluttered environments, where the robot may face several objects during the task execution. Hence, it is crucial to have systems with collision avoidance capability. In this section, we review the techniques that are suggested for this purpose. Note that there is an overlap between the materials presented in this section and Sections 3.1 and 3.2. However, we decided to devote a separate section to obstacle avoidance as it is the research question that we address in Chapter 6.

Obstacle avoidance is a classical problem in robotics and many approaches have been proposed to solve it. One may distinguish between local and global methods, depending on whether the obstacle influences the behavior either locally or everywhere. Local methods such as the Bug's algorithm (Lumelsky & Skewis, 1990), the Vector Field Histogram (Borenstein & Koren, 1991), and the Curvature-Velocity method (Simmons, 1996) offer fast response in the face of

perturbations. These are usually locally optimal and hence are not ensured to always find a feasible path.

Global methods, such as those dealt with by path planning algorithms, ensure finding a valid solution, if it exists. However, as outlined in Section 3.1, despite recent efforts at reducing the computational costs of such global searches for a feasible path, these methods cannot offer the reactivity sought for swiftly avoiding obstacles that appear suddenly. Approaches that embed the obstacle in the control law are reviewed next.

The reshaping methods aim at realtime trajectory adaptation in dynamic environments. One such method is the Elastic Band approach (Quinlan & Khatib, 1993; Brock & Khatib, 2002) in which the initial shape of the elastic band is a free path generated by a classical planner. In the presence of obstacles, this band is deformed by applying repulsive forces. The work by (Fraichard et al., 1991) also follows the same principle in which the original path is deformed locally to reflect changes in the environment topology. In these methods if the path being executed becomes infeasible due to obstacles coming into its way, the reshaping algorithm cannot be applied any longer (Yoshida & Kanehiro, 2011).

Hybrid systems that switch between local and global methods offer an interesting compromise. In (Barbehenn et al., 1994), a task is decomposed into several segments that are amenable locally. If the local approach fails, the global method is invoked. Yoshida & Kanehiro (2011) propose a reactive motion planning approach which considers both the possibility of re-planning and deformation of the path during the execution of a task. In this approach, the planner first attempts to locally modify the trajectory in the presence of an obstacle. In situations where deformation is no longer possible (i.e. the path becomes infeasible), a new feasible trajectory is re-planned. The work by Vannoy & Xiao (2008) proposes an adaptive motion planner that considers the simultaneous path and trajectory planning of high-DoF robots. This method provides multiple diverse trajectories at all times to allow instant adaptation of robot motion to newly sensed changes in the environment. The elastic roadmap approach (Y. Yang & Brock, 2007) is similar to the conventional roadmap algorithm with the difference that it allows the modification of the vertices and edges during the execution of the task, hence the roadmap always represents task-consistent motions.

In Artificial Potential Fields (Khatib, 1986) each obstacle is modeled with a repulsive force that prevents the robot from colliding with the obstacle. An appropriate repulsion force should be computed so that it repels sufficiently the trajectory away from the obstacle while avoiding to get stuck in local minima. The Attractor Dynamics approach (Iossifidis & Schöner, 2006) is another variant of the potential field method, which uses heading direction rather than the cartesian position of the vehicle. The Dynamic Potential Field (Park et al., 2008) extends the potential field principle by taking into account not just the path but also the velocity along the path. Sprunk et al. (2011) propose a

kinodynamic trajectory generation method, in which the dynamics of the robot is considered during path generation. This method uses quintic Bezier splines to specify position and orientation of the holonomic robot, and optimizes it according to a user-defined cost function.

Hoffmann et al. (2009) proposes a dynamical based approach to obstacle avoidance. This method, in essence, is very similar to the Attractor Dynamics approach in that it changes the original dynamics of motion by introducing a factor in the motion equation that stirs the motion away from the obstacle. This method is implemented to avoid point-mass objects in two and three dimensional spaces. For non-point objects, this approach requires determining a repulsion parameter that deforms the trajectory enough not to hit the obstacle.

Harmonic Potential functions (J.-O. Kim & Khosla, 1992; Feder & Slotine, 1997) were first introduced to overcome the limitation of Potential Fields. This approach takes inspiration in the description of the dynamics of (incompressible and irrotational) fluids around impenetrable obstacles. In contrast to potential field-based methods, harmonic potential-based methods are powerful in that they do not have local minima. Harmonic potentials have been used for control in numerous ways in the past few years. We mention here only the works that are closest to our method.

J.-O. Kim & Khosla (1992) were among the first groups to use harmonic potential functions to control mobile robots and in particular to control a 3-DoF arm manipulator. Feder & Slotine (1997) extended J.-O. Kim & Khosla's work to moving obstacles with constant translational and/or rotational velocities. To support multiple obstacles, they partitioned the space into regions affected by a single obstacle at most. To avoid the problem of partitioning, Waydo & Murray (2003) developed an alternative formulation using a continuous weighting factor. Similarly to (Feder & Slotine, 1997), this work only considered moving obstacles with constant velocity. A major advantage of harmonic potential functions over other potential functions is that they ensure that the target is the only attractor of the system. Unfortunately, in practice, requiring that the motions of both the robot and the obstacle follow harmonic functions may be too limiting.

3.5 Discussion and Conclusion

In this chapter, we have reviewed prominent techniques in robotics that address the problem of motion generations. Here, we aim at highlighting the current challenges in addressing the research questions considered in this thesis, and to elucidate the contribution of the present work in this regard.

We have provided an overview of planning approaches in Section 3.1. As outlined there, despite recent efforts at reducing the computational costs of such global searches for a feasible path, these methods cannot offer the reactivity sought to perform in dynamic environments. In contrast, the feedback motion

planning approaches that are presented in Section 3.2 provide such reactivity; however, these approaches are either prone to local minima, or their application is limited to planar motions, which could be quite limiting. Hence in this thesis, we develop a DS-based feedback planning technique that 1) ensures global convergence to the target (i.e. free from local minima), 2) is data-driven, and thus can be easily estimated from a set of examples, 3) can be applied to high dimensional spaces, and 4) can provide an instant adaptation to changes in environment (similarly to all feedback planning techniques).

In Section 3.3, we have presented the techniques based on imitation learning to generate robot discrete movements. We outlined that most of the trajectory-based techniques use the notion of time-indexing to describe a robot trajectory. Thus, these approaches often exhibit limited ability to generalize a task, to adapt to changes in the environment, and are very sensitive to perturbations. We then showed that DS-based modeling is an interesting alternative to trajectory-based approaches as they offer an inherent robustness to perturbations, and quite often provide a higher level of generalization ability than those based on trajectory encoding. However, as we outlined there, existing approaches either rely on some heuristics with the aim to build a locally stable estimate of nonlinear DS without any guarantee that such a model is attainable, or they depend on a (time-dependent) switching mechanism to ensure stability by shifting from an unstable nonlinear DS to a stable linear DS. We discussed that the time-dependency of such systems could yield undesired robot behaviors in the face of perturbations, and could limit their application to a small region. In this thesis, we provide a unified statistical-based framework that can 1) actually ensure both local and global stability of nonlinear DS during the training phase, without relying on any external mechanism, 2) model robot motions with autonomous nonlinear DS, hence exhibit an inherent robustness and adaptability to changes in dynamic environments, 3) encode DS using a wide variety of regression techniques, including but not limited to GMR, GPR, SVR, and LWPR, 4) only provide a high level generalization, but also allow to improve results through active learning, and 5) model time-dependent DS in case if it is essential to keep the time variable to properly model a task.

In Section 3.4, we gave an overview of different reactive obstacle avoidance approaches. Our contribution to obstacle avoidance is not intended to devise a new concept that outperforms the existing approaches. Instead we aim at providing a technique that can seamlessly integrate into the framework described above, without compromising its features such as convergence to the target, adaptability and robustness, reactivity, applicability to different models, etc. The proposed obstacle avoidance approach is similar, in spirit, to the harmonic potential functions. The main difference lies in that our approach does not require the robot to follow harmonic functions, hence it can be applied to a larger set of robot motions.

Chapter 4

Learning Reaching Movements with Dynamical Systems

> *That is what learning is. You suddenly understand something you've understood all your life, but in a new way.*
>
> Doris Lessing

So far we have described the challenges and open questions revolving around the modeling of robot reaching movements. We have presented the main motivations behind using the DS paradigm to encode robot movements, and highlighted that a planer driven by a DS enables a robot to adapt its trajectory *instantly* in a dynamically changing environment, and is inherently robust to perturbations. In this chapter we turn to the problem of how to build a stable estimate of DS from a set of demonstrations of a task.

Throughout this section, we consider demonstrations of robot motions that are performed by a human demonstrator. To avoid addressing the correspondence problem (Dautenhahn & Nehaniv, 2002), we demonstrate motions from the robot's point of view, by passively guiding the robot's arm through the task. This is done either by back-driving the robot or by teleoperating it using motion sensors (see Fig. 4.1). We hence focus on the "what to imitate" problem and derive a means to extract the generic characteristics of the dynamics of the motion. As outlined in Section 1.2.2, we assume that the relevant features of the movement, i.e. those to imitate, are the features that appear most frequently, i.e. the invariants across the demonstration. As a result, demonstrations should be such that they contain the main features of the desired task, while exploring some of the variations allowed within a neighborhood around the space covered by the demonstrations.

We begin this chapter by introducing our formalism in Section 4.1. There, we also present a schematic of the control flow showing how we integrate a DS planar into a robot's built-in controller. Then, in Section 4.2, we delineate through a number of examples the main challenges in learning a stable estimate of nonlinear DS, and showcase why such learning is non-trivial even for sim-

Figure 4.1: Demonstrating motions by teleoperating a robot using motion sensors (**left**) or by back-driving it (**right**).

ple motions. In Section 4.3, we provide a more comprehensive description of Gaussian mixture modeling as it is the main approach that we exploit to encode robot motions.

In Section 4.4, we present our first attempt to build a stable model of autonomous DS through an iterative approach called *Binary Merging (BM)*. This method formulates DS as a mixture of Gaussian functions, and proceeds by minimizing the number of Gaussian functions required for achieving both *local asymptotic stability* at the target and accuracy in estimating demonstrations. This work was published in (Khansari-Zadeh & Billard, 2010a), and part of the material from this publication is collected here.

In Section 4.5, we formulate the problem of building an estimate of autonomous DS as a constrained optimization problem. We then propose a learning algorithm, called *Stable Estimator of Dynamical Systems (SEDS)*, that maximizes the accuracy in estimation of the DS while ensuring its global asymptotic stability at the target. This work was published in (Khansari-Zadeh & Billard, 2010b, 2011), and all the material from these publications is collected here.

We then further extend this approach in Section 4.6 and introduce *SEDS-II*, which can be used to ensure global asymptotic stability of DS-based motions independently of the choice of the regression technique. This approach can also provide the possibility of online learning and can estimate motions that are represented with both autonomous and non-autonomous DS.

In Section 4.7, we compare the performance of the above approaches against each other, and four of the best performing regression methods to date namely GPR, GMR, LWPR, and SVR. Through this comparison, we highlight the pros and cons of each method, which could eventually help the reader to choose the best approach depending on the task at hand. We summarize and conclude this chapter in Section 4.8.

It should be noted that in (Gribovskaya et al., 2010), we proposed a numerical approach to build iteratively a *locally* stable estimate of nonlinear DS. This approach starts with a single Gaussian function that models a stable linear

DS. This initialization is a poor proxy of the demonstration trajectories. The method then iteratively adds a new Gaussian function, re-train at each step the complete mixture (except for the first Gaussian function) and test the stability numerically, by creating a mesh of fixed volume around the demonstrated trajectories. If all trajectories initiated on the points of the mesh converge to the attractor, the system is said to be locally stable in the volume, which represents the region of attraction.

While this approach worked well in practice, it suffered from three main limitations. It did not ensure to find a good estimate of the true region of attraction. The method is computationally intensive, growing exponentially with the dimension of the system. Finally and most importantly, there is no guarantee that the region of attraction is stable, as sole a finite set of points in that region are tested.

In contrast, in the three approaches that are presented in this section, we develop a formal analysis of stability and formulate explicit constraints on the parameters of the DS to ensure its asymptotic stability at the target. Furthermore the two approaches, called SEDS and SEDS-II, can ensure global asymptotic stability at a unique target point, hence providing a large domain of applicability. Additionally, these approaches benefit from having a very fast training algorithm without any need to perform computationally expensive numerical analysis.

Related publications:

- S.M. Khansari Zadeh and A. Billard (2011), Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models, *IEEE Transaction on Robotics*, 27(5), p. 943–957.
- S.M. Khansari Zadeh and A. Billard (2010a), BM: An Iterative Method to Learn Stable Non-Linear Dynamical Systems with Gaussian Mixture Models, *In proceedings of the International Conference on Robotics and Automation (ICRA)*, p. 2381–2388.
- S.M. Khansari Zadeh and A. Billard (2010b), Imitation learning of Globally Stable Non-Linear Point-to-Point Robot Motions using Nonlinear Programming, *In proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 2676–2683.

4.1 Formalism

We formulate the encoding of point-to-point motions as a control law driven by autonomous dynamical systems. Consider a state variable $\boldsymbol{\xi} \in \mathbb{R}^d$ that can be used to *unambiguously* define a discrete motion of a robotic system (e.g. $\boldsymbol{\xi}$ could be a robot's joint angles, the position of an arm's end-effector in the Cartesian space, etc). Let the set of N given demonstrations $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$ be instances of a global motion model governed by a first order autonomous Ordinary Differential Equation (ODE):

$$\dot{\boldsymbol{\xi}} = \mathbf{f}(\boldsymbol{\xi}) + \epsilon \tag{4.1}$$

where $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a nonlinear continuous and continuously differentiable function with a single equilibrium point $\dot{\boldsymbol{\xi}}^* = \mathbf{f}(\boldsymbol{\xi}^*) = \mathbf{0}$ and ϵ represents a zero mean additive Gaussian noise. The noise term ϵ encapsulates both inaccuracies in sensor measurements and errors resulting from imperfect demonstrations. Note that in our formalism, we assume that the demonstrations are consistent according to Eq. (4.1), and hence that demonstrations passing through the same point should do so with roughly the same speed profile.

The function $\mathbf{f}(\boldsymbol{\xi})$ can be described by a set of parameters $\boldsymbol{\theta}$, in which the optimal values of $\boldsymbol{\theta}$ can be obtained based on the set of demonstrations using different statistical approaches (for instance see the methods described in Section 2.2). We will further denote the obtained noise-free estimate of $\mathbf{f}$ from the statistical modeling with $\boldsymbol{f}$ throughout this thesis. Our noise-free estimate will thus be:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}; \boldsymbol{\theta}) \tag{4.2}$$

Given an initial point $\boldsymbol{\xi}^0 \in \mathbb{R}^d$, the evolution of motion can be computed by integrating Eq. (4.2) through time:

$$\boldsymbol{\xi}(t) = \int_0^t \boldsymbol{f}(\boldsymbol{\xi}; \boldsymbol{\theta}) dt \tag{4.3}$$

Analytical computation of the above integral is usually non-trivial, especially for complex multi-dimensional DS. Alternatively, Eq. (4.3) can be estimated numerically with:

$$\boldsymbol{\xi}^t = \boldsymbol{\xi}^{t-1} + \boldsymbol{f}(\boldsymbol{\xi}; \boldsymbol{\theta}) \delta t \tag{4.4}$$

where δt is the integration time step and t is a positive integer. The result of Eq. (4.4) converges to Eq. (4.3) as $\delta t \rightarrow 0$.

Two observations follow from formalizing our problem using Eq. (4.2): 1) The control law given by Eq. (4.2) will generate trajectories that do not intersect, even if the original demonstrations did intersect; 2) The motion of the system is uniquely determined by its state $\boldsymbol{\xi}$. The choice of state variable $\boldsymbol{\xi}$ is hence crucial. For instance, if one wishes to represent trajectories that intersect in the Cartesian space, one should encode both position and velocity in $\boldsymbol{\xi}$, i.e. $\boldsymbol{\xi} = [\boldsymbol{x}; \dot{\boldsymbol{x}}]$.

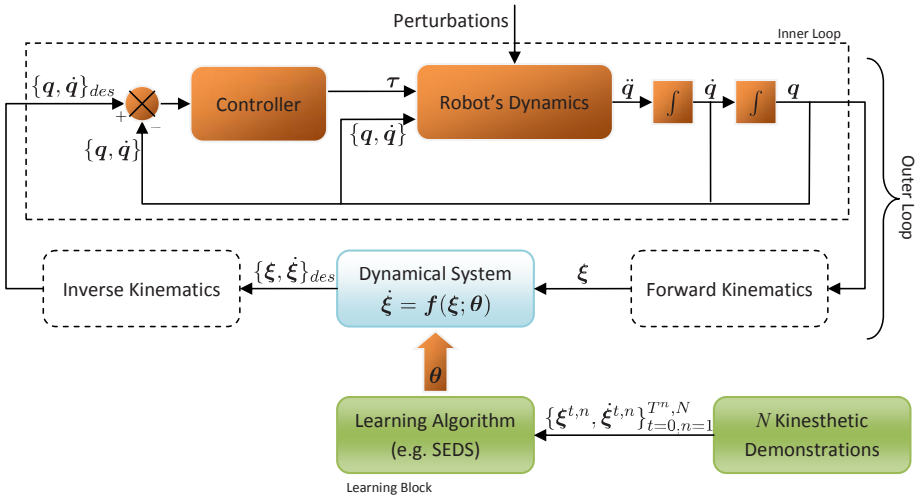


Figure 4.2: A typical system's architecture illustrating the control flow in a robotic system as considered throughout this thesis. In this graph, $\boldsymbol{q}$, $\boldsymbol{\tau}$, and $\boldsymbol{\xi}$ correspond to the robot's joint angles, joint torques, and the state variables describing the robot motion, respectively. The system's architecture is composed of two loops: the inner loop representing the robot's dynamics and a low level controller, and an outer loop defining the desired motion at each time step. The learning block is used to infer the parameters of motion $\boldsymbol{\theta}$ from demonstrations.

Throughout this thesis we choose to represent a motion in kinematic coordinates system (i.e. the Cartesian or $\mathcal{C}$-space), and assume that there exists a low-level controller that converts kinematic variables into motor commands (e.g. force or torque). Figure 4.2 shows a schematic of the control flow. The whole system's architecture can be decomposed into two loops. The inner loop consists of a controller generating the required commands to follow the desired motion and a system block to model the dynamics of the robot. Here $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are the robot's joint angles and their first and second time derivatives. Motor commands are denoted by $\boldsymbol{\tau}$. The outer loop specifies the next desired position and velocity of the motion with respect to the current status of the robot. An inverse kinematic block may also be considered in the outer loop to transfer the desired trajectory from the Cartesian to the $\mathcal{C}$-space (this block is not necessary if the motion is already specified in the $\mathcal{C}$-space).

In this control architecture, both the inner and outer loops should be stable. The stability of the inner loop requires the system to be Input-to-State Stable (ISS) (Sontag, 2008), i.e. the output of the inner loop should remain bounded for a bounded input. The stability of the outer loop should be ensured when learning the DS model of the motion. The learning block refers to the procedure that determines a stable estimate of the DS to be used as the outer loop control. Throughout this thesis, we choose the control approaches that were presented in Section 2.4 to make the inner loop ISS. Hence we focus our efforts on designing a learning block that ensures stability of the outer loop controller. Learning is data-driven and uses a set of demonstrated trajectories to determine the parameters $\boldsymbol{\theta}$ of the DS given by Eq. (4.2).

4.2 Challenges

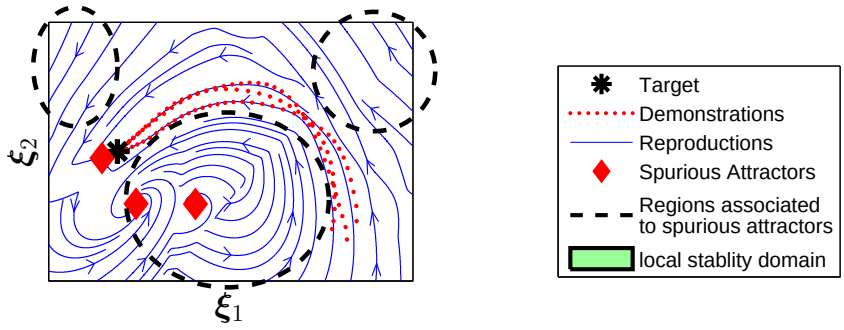
In Section 2.2, we presented a number of techniques to build an estimate of nonlinear DS from a set of demonstrations. As we have highlighted, because all these methods do not optimize under the constraint of making the system stable at the target, they are not guaranteed to result in a stable estimate of the motion. In practice, they fail to ensure global stability and they also rarely ensure local stability of $\boldsymbol{f}$. Such estimates of the motion may hence converge to spurious attractors or miss the target (diverging/unstable behavior) even when estimating simple motions such as motions in the plane. This is due to the fact that there is yet no generic theoretical solution to ensuring stability of arbitrary nonlinear autonomous DS (Slotine & Li, 1991).

Figure 4.3 illustrates an example of unstable estimation of a 2D motion using three different regression techniques. The input and output of the DS are the Cartesian position and velocity of a virtual point robot on a horizontal plane, respectively. The demonstrations are collected from pen input using a Tablet-PC. Figure 4.3a represents the stability analysis of the DS when it is learned with GMR. Here in the narrow regions around demonstrations, the trajectories converge to a spurious attractor just next to the target. In other parts of the space, they either converge to other spurious attractors far from the target or completely diverge from it. Figure 4.3b shows the obtained results from LWPR. All trajectories inside the black boundaries converge to a spurious attractor. Outside of these boundaries, the velocity is always zero (a region of spurious attractors) hence a motion stops once it reaches these boundaries or it does not move when it initializes there. Regarding Fig. 4.3c, while for GPR trajectories converge to the target in a narrow area close to the demonstrations, they are attracted to spurious attractors outside that region.

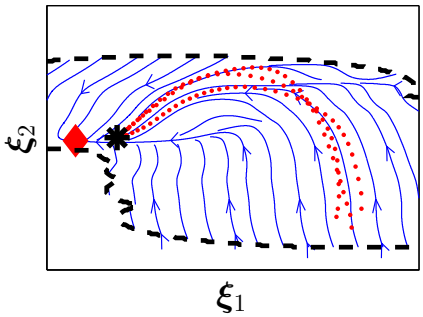
In all these examples, regions of attractions are usually very close to demonstrations and thus should be carefully avoided. However, the critical concern is that there is not a generic theoretical solution to determine beforehand whether a trajectory will lead to a spurious attractor, to infinity, or to the desired attractor. Thus, it is necessary to conduct numerical stability analysis to locate the region of attraction of the desired target which may never exist, or be very narrow. Figures 4.3d to 4.3g show the obtained results using the methods that will be presented in this chapter. As can be seen, trajectories generated from these approaches can accurately follow the demonstrations, while their convergence to the target is ensured.

4.3 Multivariate Regression through GMM

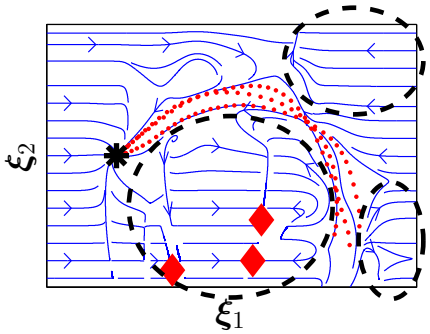
As outlined before, we use a probabilistic framework and model $\boldsymbol{f}$ via a finite mixture of Gaussian functions. We have already provided a brief description of



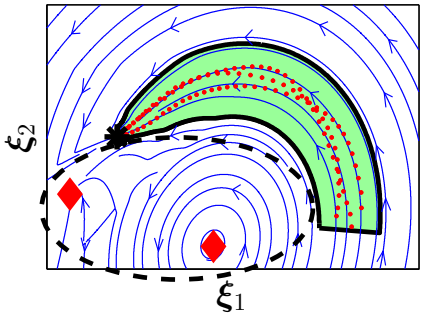
(a) GMR



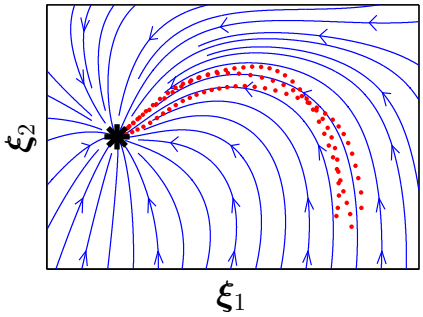
(b) LWPR



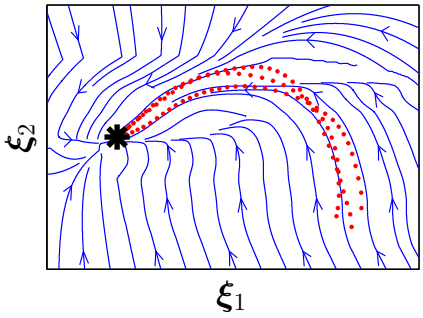
(c) GPR



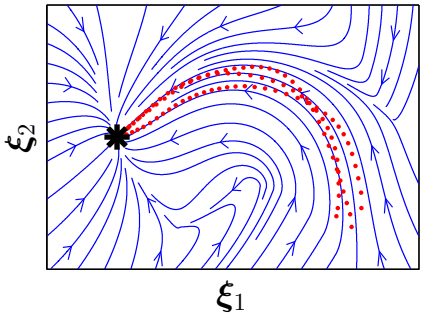
(d) BM (locally asymptotically stable)



(e) SEDS (globally asymptotically stable)



(f) SEDS-II with LWPR encoding (globally asymptotically stable)



(g) SEDS-II with GPR encoding (globally asymptotically stable)

Figure 4.3: An example of two-dimensional dynamics learned from three demonstrations using six different methods: GMR, LWPR, GPR, BM, SEDS, and SEDS-II with LWPR and GPR formulations. All reproductions were generated in simulation. For further information please refer to Section 4.2.

GMM and GMR in Section 2.2.1. Here we explain this approach in more details as it is the key regression technique that we adopt in this thesis.

Mixture modeling is a popular approach for density approximation (McLachlan & Peel, 2000), and it allows a user to define an appropriate model through a tradeoff between model complexity and variations of the available training data. Mixture modeling is a method, that builds a coarse representation of the data density through a fixed number (usually lower than 10) of mixture components. An optimal number of components can be found using various methods, such as the Bayesian Information Criterion (BIC) (Schwarz, 1978), the Akaike information criterion (AIC) (Akaike, 1974), the deviance information criterion (DIC) (Spiegelhalter et al., 2002), that penalize large increase in the number of parameters when it only offers a small gain in the likelihood of the model.

By estimating $\boldsymbol{f}$ via a finite mixture of Gaussian functions, the unknown parameters of $\boldsymbol{f}$ become the prior π^k, the mean $\boldsymbol{\mu}^k$ and the covariance matrices $\boldsymbol{\Sigma}^k$ of the $k = 1..K$ Gaussian functions (i.e. $\boldsymbol{\theta}^k = \{\pi^k, \boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k\}$ and $\boldsymbol{\theta} = \{\boldsymbol{\theta}^1..\boldsymbol{\theta}^K\}$). The mean and the covariance matrix of each Gaussian function are defined by:

$$\boldsymbol{\mu}^k = \begin{pmatrix} \boldsymbol{\mu}^k_{\boldsymbol{\xi}} \\ \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}} \end{pmatrix} \quad \& \quad \boldsymbol{\Sigma}^k = \begin{pmatrix} \boldsymbol{\Sigma}^k_{\boldsymbol{\xi}} & \boldsymbol{\Sigma}^k_{\boldsymbol{\xi}\dot{\boldsymbol{\xi}}} \\ \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} & \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}} \end{pmatrix} \qquad \forall k \in 1..K \tag{4.5}$$

Given a set of N demonstrations $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$, each recorded point in the trajectories $[\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]$ is associated with a probability density function $\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}])$:

$$\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]; \boldsymbol{\theta}) = \sum_{k=1}^{K} \mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k) \qquad \begin{cases} \forall n \in 1..N \\ t \in 0..T^n \end{cases} \tag{4.6}$$

where $\mathcal{P}(k) = \pi^k$ is the prior and $\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)$ is the conditional probability density function given by:

$$\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k) = \mathcal{N}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]; \boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k) = \frac{1}{\sqrt{(2\pi)^{2d}|\boldsymbol{\Sigma}^k|}} e^{-\frac{1}{2}([\boldsymbol{\xi}^{t,n};\dot{\boldsymbol{\xi}}^{t,n}]-\boldsymbol{\mu}^k)^T(\boldsymbol{\Sigma}^k)^{-1}([\boldsymbol{\xi}^{t,n};\dot{\boldsymbol{\xi}}^{t,n}]-\boldsymbol{\mu}^k)} \tag{4.7}$$

Taking the posterior mean estimate of $\mathcal{P}(\dot{\boldsymbol{\xi}}|\boldsymbol{\xi})$ yields:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}) = \sum_{k=1}^{K} \frac{\mathcal{P}(k)\mathcal{P}(\boldsymbol{\xi}|k)}{\sum_{i=1}^{K} \mathcal{P}(i)\mathcal{P}(\boldsymbol{\xi}|i)} \left(\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1}(\boldsymbol{\xi} - \boldsymbol{\mu}^k_{\boldsymbol{\xi}}) + \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}\right) \tag{4.8}$$

The notation of Eq. (4.8) can be simplified through a change of variable. Let us define:

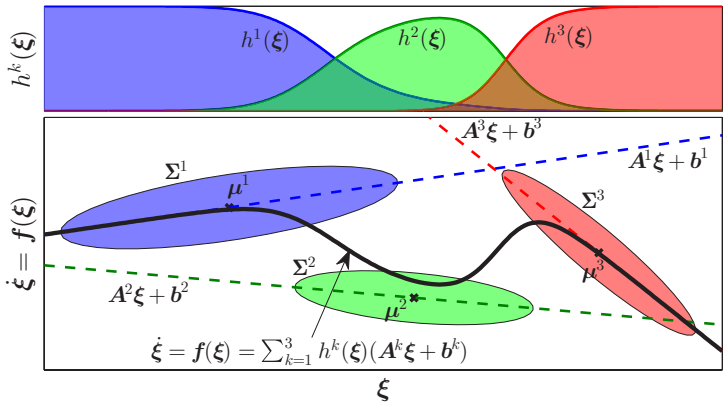


Figure 4.4: Illustration of parameters defined in Eq. (4.9) and their effects on $\boldsymbol{f}(\boldsymbol{\xi})$ for a 1D model constructed with 3 Gaussians. Please refer to the text for further information.

$$\begin{cases} \boldsymbol{A}^k = \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \\ \boldsymbol{b}^k = \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}} - \boldsymbol{A}^k\boldsymbol{\mu}^k_{\boldsymbol{\xi}} \\ h^k(\boldsymbol{\xi}) = \frac{\mathcal{P}(k)\mathcal{P}(\boldsymbol{\xi}|k)}{\sum_{i=1}^{K}\mathcal{P}(i)\mathcal{P}(\boldsymbol{\xi}|i)} \end{cases} \tag{4.9}$$

Substituting Eq. (4.9) into Eq. (4.8) yields:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}) = \sum_{k=1}^{K} h^k(\boldsymbol{\xi})(\boldsymbol{A}^k\boldsymbol{\xi} + \boldsymbol{b}^k) \tag{4.10}$$

Note that $\boldsymbol{f}$ is now expressed as a nonlinear sum of linear dynamical systems. Such a rewriting will prove useful to study the effect of each Gaussian in the final reproduction. Figure 4.4 illustrates the parameters of Eq. (4.9) and their effects on Eq. (4.10) for a one-dimensional (1D) model constructed with 3 Gaussian functions. Here, each linear dynamics $\boldsymbol{A}^k\boldsymbol{\xi}+\boldsymbol{b}^k$ corresponds to a line that passes through the centers $\boldsymbol{\mu}^k$ with slope $\boldsymbol{A}^k$. The nonlinear weighting terms $h^k(\boldsymbol{\xi})$ in Eq. (4.10), where $0 < h^k(\boldsymbol{\xi}) \leq 1$, give a measure of the relative influence of each Gaussian locally: the more variance (the less accurate the demonstrations), the less influence.

Observe that due to the nonlinear weighting terms $h^k(\boldsymbol{\xi})$, the resulting function $\boldsymbol{f}(\boldsymbol{\xi})$ is nonlinear and is flexible enough to model a wide variety of motions. If one estimates this mixture using classical methods such as EM, one cannot guarantee that the system will be asymptotically stable. The resulting nonlinear model $\boldsymbol{f}(\boldsymbol{\xi})$ usually contains several spurious attractors or limit cycles even for a simple 2D model (see Fig. 4.3). Beware that the intuition that the nonlinear function $\boldsymbol{f}(\boldsymbol{\xi})$ should be stable if all eigenvalues of matrices $\boldsymbol{A}^k$, $k = 1..K$, have strictly negative real parts is not true. Here is a simple example in 2D that illustrates why this is not the case and also why estimating stability of nonlinear DS even in 2D is non-trivial.

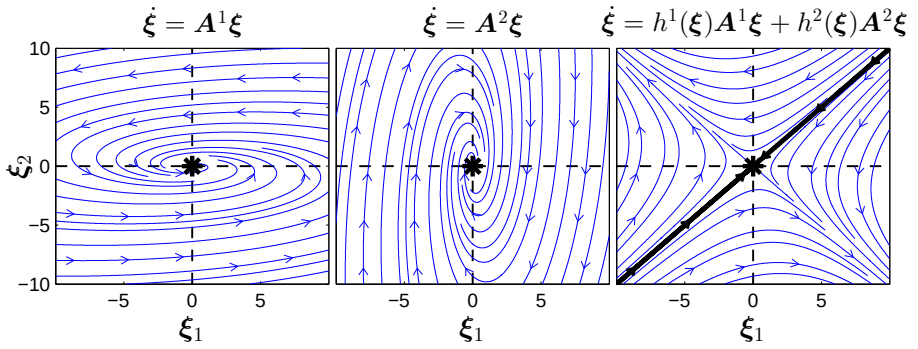


Figure 4.5: While each of the subsystem $\dot{\boldsymbol{\xi}} = \boldsymbol{A}^1\boldsymbol{\xi}$ (left) and $\dot{\boldsymbol{\xi}} = \boldsymbol{A}^2\boldsymbol{\xi}$ (center) is asymptotically stable at the origin, a nonlinear weighted sum of these systems $\dot{\boldsymbol{\xi}} = h^1(\boldsymbol{\xi})\boldsymbol{A}^1\boldsymbol{\xi} + h^2(\boldsymbol{\xi})\boldsymbol{A}^2\boldsymbol{\xi}$ may become unstable (right). Here the system remains stable only for points on the line $\boldsymbol{\xi}_2 = \boldsymbol{\xi}_1$ (drawn in black).

Example: Consider the parameters of a model with two Gaussian functions to be:

$$\begin{cases} \boldsymbol{\Sigma}^1_{\boldsymbol{\xi}} = \boldsymbol{\Sigma}^2_{\boldsymbol{\xi}} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \\ \boldsymbol{\Sigma}^1_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}}} = \begin{bmatrix} -3 & -30 \\ 3 & -3 \end{bmatrix}, \ \boldsymbol{\Sigma}^2_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}}} = \begin{bmatrix} -3 & 3 \\ -30 & -3 \end{bmatrix} \\ \boldsymbol{\mu}^1_{\boldsymbol{\xi}} = \boldsymbol{\mu}^2_{\boldsymbol{\xi}} = \boldsymbol{\mu}^1_{\dot{\boldsymbol{\xi}}} = \boldsymbol{\mu}^2_{\dot{\boldsymbol{\xi}}} = \mathbf{0} \end{cases} \tag{4.11}$$

Using Eq. (4.9) we have:

$$\begin{cases} \boldsymbol{A}^1 = \begin{bmatrix} -1 & -10 \\ 1 & -1 \end{bmatrix}, \ \boldsymbol{A}^2 = \begin{bmatrix} -1 & 1 \\ -10 & -1 \end{bmatrix} \\ \boldsymbol{b}^1 = \boldsymbol{b}^2 = \mathbf{0} \end{cases} \tag{4.12}$$

The eigenvalues of the two matrices $\boldsymbol{A}^1$ and $\boldsymbol{A}^2$ are complex with values $-1 \pm 3.16i$. Hence, each matrix determines a stable system. However, the nonlinear combination of the two matrices as per Eq. (4.10) is stable only when $\boldsymbol{\xi}_2 = \boldsymbol{\xi}_1$, and is unstable in $\mathbb{R}^d \setminus \{(\boldsymbol{\xi}_2, \boldsymbol{\xi}_1) | \boldsymbol{\xi}_2 = \boldsymbol{\xi}_1\}$ (see Fig. 4.5).

4.4 Binary Merging

In this section, we provide a set of stability conditions that can be used to ensure local asymptotic stability of $\boldsymbol{f}$ when it is formulated with a mixture of Gaussian functions. We then propose a learning procedure, called Binary Merging (BM), that tackles the problem of estimating an unknown nonlinear DS from a few demonstrations while ensuring its local stability at the target based on the provided stability conditions. BM builds an estimate of $\boldsymbol{f}$ by minimizing the

number of Gaussian functions required for achieving both asymptotic stability at the target and high accuracy in estimating the dynamics of motion.

This section is structured as follows. In Section 4.4.1 we develop conditions for ensuring local stability of $\boldsymbol{f}$. In Section 4.4.2 we describe the BM learning algorithm. In Section 4.4.3, we present the experimental validation of the method, and finally we devote Section 4.4.4 to discussion and conclusion.

4.4.1 Stability Analysis

Without loss of generality, assume that the target point $\boldsymbol{\xi}^*$ is located at the origin, i.e. $\boldsymbol{f}(\boldsymbol{\xi}^*) = \boldsymbol{f}(\mathbf{0}) = \mathbf{0}$. Let $D \subset \mathbb{R}^d$ be a region that covers entirely the part of the state space spanned by the demonstrations and includes the origin:

Definition 4.1 *Consider a scalar $\delta > 0$:*

$$\delta = \min(\mathcal{P}(\boldsymbol{\xi}^{t,n})) \qquad \forall t = 0..T^n, n = 1..N \tag{4.13}$$

where $\mathcal{P}(\boldsymbol{\xi})$ is the probability of $\boldsymbol{\xi}$ estimated from Eq. (4.6) and $\{\boldsymbol{\xi}^{t,n}\}_{t=0,n=1}^{T^n,N}$ are demonstration trajectories. There exist a scalar $0 < \alpha \leq 1$ such that the region

$$D = \{\, \boldsymbol{\xi} \in \mathbb{R}^d \;\; : \;\; \mathcal{P}(\boldsymbol{\xi}) \geq \alpha\delta \} \tag{4.14}$$

defines a connected partition[1] of the state space that comprises all the training datapoints, including the origin.

This definition of δ ensures that all training datapoints are included in D. The scalar α is also required to obtain a connected region. To study the stability of $\boldsymbol{f}$, we partition D into K pairwise disjoint continuous subregions Ω^k via hyperplanes $\boldsymbol{\Phi}^k$, $k = 1..K-1$.

Definition 4.2 *Consider a finite set of $k = 1..K$ Gaussian functions numbered $\mathcal{N}^1$ through $\mathcal{N}^K$. Let $\boldsymbol{\mu}^k$ and $\boldsymbol{\Sigma}^k$ be, respectively, the mean and covariance matrix of the Gaussian $\mathcal{N}^k$ as given by Eq. (4.5). Let the vector $\boldsymbol{\upsilon}^k$ be the eigenvector of $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k$ forming the smallest angle with $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$ (i.e. $\boldsymbol{\upsilon}^k$ is the eigenvector pointing towards the direction of motion). Then $\boldsymbol{\Phi}^k$ is the hyperplane through $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$ and normal to $\boldsymbol{\upsilon}^k$:*

$$\boldsymbol{\Phi}^k : (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T \cdot \boldsymbol{\upsilon}^k = 0 \qquad \forall k \in 1..K-1 \tag{4.15}$$

Definition 4.3 *The state-space domain D is partitioned into K pairwise disjoint continuous subregions Ω^k, $\bigcup_{k=1}^{K} \Omega^k = D$, $\Omega^k \cap \Omega^j = \emptyset$, $\forall k, j \in 1..K$ and $j \neq k$. Each subregion Ω^k is a part of D that is defined by:*[2]

[1] A partition D is connected if any two points in D can be connected by a curve lying completely within D.

[2] Here we assume that we obtain K pairwise disjoint partitions after applying Eq. (4.16). This assumption is essential in order to ensure stability of the system for the method that is presented in this section.

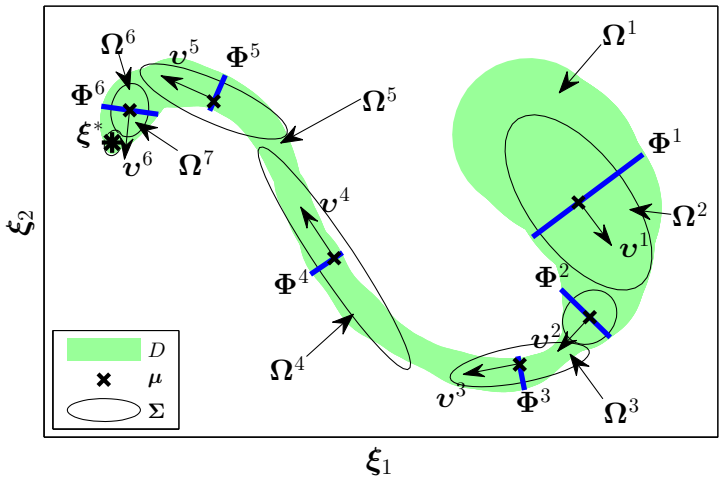


Figure 4.6: Representation of subdomains $\boldsymbol{\Omega}^k(\boldsymbol{\xi})$ for a sample 2D model. Please see the text for further information.

$$\Omega^k = \hat{\Omega}^k \bigcap D \qquad \forall k \in 1..K \tag{4.16}$$

where

$$\begin{cases} \hat{\Omega}^1 = \{\, \boldsymbol{\xi} \in D \;:\; (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^1)^T \cdot \boldsymbol{v}^1 \leq 0 \} \\ \hat{\Omega}^k = \{\, \boldsymbol{\xi} \in D \;:\; (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k-1})^T \cdot \boldsymbol{v}^{k-1} > 0, \quad (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k})^T \cdot \boldsymbol{v}^{k} \leq 0 \} \quad \forall k \in 2..K-1 \\ \hat{\Omega}^K = \{\, \boldsymbol{\xi} \in D \;:\; (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{K-1})^T \cdot \boldsymbol{v}^{K-1} > 0 \} \end{cases}$$

Figure 4.6 illustrates an example of using Definition 4.3 to parting D into $K = 7$ pairwise disjoint continuous subregions. In each subdomain $\Omega^k \subset D$, $k = 2..K$, we truncate the estimate given by Eq. (4.10) so that the dynamics are driven solely by the two dominant Gaussian functions $\mathcal{N}^{k-1}$ and $\mathcal{N}^k$. Because $h^k(\boldsymbol{\xi})$ decays asymptotically as one moves away from the center of the associated Gaussian, the effect of truncating the influence of non-adjacent Gaussian functions is in practice negligible[3]. Note that we only use $\mathcal{N}^1$ to estimate $\boldsymbol{f}$ for all points in Ω^1 as by construction we set it to be the only dominant Gaussian in this partition (see Definition 4.3 and Fig. 4.6). Thus we have:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}) = \begin{cases} \boldsymbol{A}^1\boldsymbol{\xi} + \boldsymbol{b}^1 & \forall \boldsymbol{\xi} \in \Omega^1 \\ \\ h^{k-1}(\boldsymbol{\xi})(\boldsymbol{A}^{k-1}\boldsymbol{\xi} + \boldsymbol{b}^{k-1}) + h^k(\boldsymbol{\xi})(\boldsymbol{A}^k\boldsymbol{\xi} + \boldsymbol{b}^k) & \forall \boldsymbol{\xi} \in \Omega^k, k \in 2..K \end{cases} \tag{4.17}$$

[3] This is especially true if the distance between the centers of the Gaussian functions is much larger than the variance of each Gaussian. Note that this formulation makes $\dot{\boldsymbol{\xi}}$ discontinuous at the boundaries between the subregions. This however does not affect the proof of Theorem 4.1, since the stability conditions given by Eq. (4.18) do not require continuity at the boundaries (Pettersson & Lennartson, 1997; Borne & Dieulot, 2005).

□

Theorem 4.1 *Assume that the state trajectory evolves according to Eq. (4.17). Then the origin of Eq. (4.17) is asymptotically stable in D if the parameters of $\boldsymbol{f}$ (i.e. $\boldsymbol{\mu}^k$ and $\boldsymbol{\Sigma}^k$, $\forall k = 1..K$, $K > 1$) are constructed such that:*

$$\begin{cases} \boldsymbol{\mu}_{\boldsymbol{\xi}}^K = \boldsymbol{\xi}^* = \mathbf{0} \\ \boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^K = -\frac{\mathcal{P}(\mathbf{0}|K-1)}{\mathcal{P}(\mathbf{0}|K)}\big(\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^{K-1} - \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^{K-1}(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{K-1})^{-1}\boldsymbol{\mu}_{\boldsymbol{\xi}}^{K-1}\big) \\ \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^K(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^K)^{-1} + (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^K)^{-1}(\boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^K)^T \prec 0 \end{cases} \tag{4.18a}$$

$$\begin{cases} (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^1)^T(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^1)^{-1}\dot{\boldsymbol{\xi}} < 0 \qquad \forall \boldsymbol{\xi} \in \Omega^1 \\ (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k-1})^T(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{k-1})^{-1}\dot{\boldsymbol{\xi}} > (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1}\dot{\boldsymbol{\xi}} \qquad \begin{cases} \forall \boldsymbol{\xi} \in \Omega^k \\ \forall \boldsymbol{\xi} \neq \mathbf{0} \\ k = 2..K \end{cases} \end{cases} \tag{4.18b}$$

$$(\boldsymbol{\upsilon}^k)^T\dot{\boldsymbol{\xi}} > 0 \qquad \forall \boldsymbol{\xi} \in \boldsymbol{\Phi}^k,\ \forall k \in 1..K-1 \tag{4.18c}$$

$$D \text{ is an invariant set} \tag{4.18d}$$

where $. \prec 0$ refers to the negative definiteness of a matrix[4]*.*

Proof: See Appendix A.1.

■

To elaborate more, condition (4.18a) puts a constraint on Eq. (4.17) to force the origin to be an equilibrium point. Condition (4.18b) defines criteria to ensure that starting from any point $\boldsymbol{\xi} \in D$, the energy of the system (i.e. the Lyapunov function) decreases as the motion evolves. Condition (4.18c) ensures the transition of the motion from one partition to another partition at the boundaries $\boldsymbol{\Phi}^k$. Condition (4.18d) is necessary to verify that the generated trajectories from the DS do not leave the stable region. Putting together conditions (4.18a)-(4.18d), the system becomes locally asymptotically stable at the origin in the region defined by D.

4.4.2 Learning Algorithm

Section 4.4.1 provided us with conditions whereby the estimate, produced according to our state evolution paradigm given by Eq. (4.17), is asymptotically stable at the origin in D. It remains now to determine a procedure by which we can construct a mixture of Gaussian functions to satisfy the conditions given by Eq. (4.18). Not only should the estimate be stable according to our earlier definition, but it should also provide an accurate estimate of the overall

[4] A $d \times d$ real symmetric matrix $\boldsymbol{A}$ is positive definite if $\boldsymbol{\xi}^T\boldsymbol{A}\boldsymbol{\xi} > 0$ for all $\boldsymbol{\xi} \in \mathbb{R}^d \setminus \mathbf{0}$, where $\boldsymbol{\xi}^T$ denotes the transpose of $\boldsymbol{\xi}$. Conversely $\boldsymbol{A}$ is negative definite if $\boldsymbol{\xi}^T\boldsymbol{A}\boldsymbol{\xi} < 0$. For a non-symmetric matrix, $\boldsymbol{A}$ is positive (negative) definite if and only if its symmetric part $\tilde{\boldsymbol{A}} = (\boldsymbol{A} + \boldsymbol{A}^T)/2$ is positive (negative) definite.

Algorithm 4.1 Binary Merging (BM)

Input: $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$, r, q, and e_{max}
Initialization:
Transfer $\boldsymbol{\xi}^*$ to the origin
Apply sample alignment to get N demonstrations, all of length T
$K \leftarrow T$
Initialize a GMM $\boldsymbol{\theta} = \{\boldsymbol{\theta}^1..\boldsymbol{\theta}^K\}$ using Eq. (4.21)
Main Body:
while $K > 1$ **and** further merging is possible **do**
  Backup the previous model $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$ and dataset $\tilde{\boldsymbol{\Xi}} \leftarrow \boldsymbol{\Xi}$
  Randomly select an index $k \in 1..K-1$
  Replace $\boldsymbol{\theta}^k \leftarrow \{\boldsymbol{\theta}^k + \boldsymbol{\theta}^{k+1}\}$ by merging $\boldsymbol{\theta}^k$ and its adjacent $\boldsymbol{\theta}^{k+1}$
  Update the dataset of the k-th Gaussian: $\boldsymbol{\Xi}^k \leftarrow [\boldsymbol{\Xi}^k, \boldsymbol{\Xi}^{k+1}]$
  Remove $\boldsymbol{\theta}^{k+1}$ and correct the numbering of Gaussians $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i+1}$ and the datasets $\boldsymbol{\Xi}^i = \boldsymbol{\Xi}^{i+1}$, $\forall i \in k+1..K-1$
  $K \leftarrow K-1$
  Check stability conditions using Algorithm 4.2
  if conditions of Theorem 4.1 are violated **or if** Eq. (4.20) is no longer satisfied **then**
    Recover the previous model $\boldsymbol{\theta} \leftarrow \tilde{\boldsymbol{\theta}}$ and dataset $\boldsymbol{\Xi} \leftarrow \tilde{\boldsymbol{\Xi}}$
    Set $K \leftarrow K+1$
  end if
end while
Output: $\boldsymbol{\theta} = \{\boldsymbol{\theta}^1..\boldsymbol{\theta}^K\}$ and D

dynamics. We evaluate the latter through a measure of accuracy with which $\boldsymbol{f}$ approximates the demonstrations. This can be quantified by measuring the discrepancy between the direction and amplitude of the estimated and observed velocity vectors for all the training points $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$:

$$\bar{e} = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{T^n}\sum_{t=0}^{T^n} r\Big(1 - \frac{(\dot{\boldsymbol{\xi}}^{t,n})^T \boldsymbol{f}(\boldsymbol{\xi}^{t,n})}{\|\dot{\boldsymbol{\xi}}^{t,n}\|\|\boldsymbol{f}(\boldsymbol{\xi}^{t,n})\| + \epsilon}\Big)^2 + \cdots \right. \\ \left. q\frac{\big(\dot{\boldsymbol{\xi}}^{t,n} - \boldsymbol{f}(\boldsymbol{\xi}^{t,n})\big)^T\big(\dot{\boldsymbol{\xi}}^{t,n} - \boldsymbol{f}(\boldsymbol{\xi}^{t,n})\big)}{\|\dot{\boldsymbol{\xi}}^{t,n}\|\|\dot{\boldsymbol{\xi}}^{t,n}\| + \epsilon}\right)^{0.5} \tag{4.19}$$

where r and q are positive scalars that weight the relative influence of each factor, and ϵ is a very small positive scalar. Given a maximal acceptable error e_{max}, estimates of the dynamics are accurate if

$$\bar{e} \leq e_{max} \tag{4.20}$$

We now present our learning approach, called Binary Merging (BM), that can build a stable estimate of $\boldsymbol{f}$. BM proceeds in two steps. First it initializes a model with the maximum possible number of Gaussian functions. Then it incrementally reduces the number of Gaussian functions to a minimum number (locally), which satisfies the stability criteria while keeping the error of the estimates below a certain level. Algorithm 4.1 shows the pseudocode of the BM procedure. Next, we briefly explain the main steps.

Initialization: First, demonstration trajectories are aligned using a *sample alignment method* (Myers & Rabiner, 1981). These trajectories usually differ in length, as they may have been performed at different speeds. With sample alignment, demonstrations are aligned such that data points with the same time stamp have the most similarity based on a specified fitness function. The output of sample alignment is N demonstrations all of length T. Sample alignment differs from Dynamic Time Warping in that it does not distort the temporal transitions between datapoints in a demonstration. Figures 4.7a and 4.7b illustrate the sample alignment procedure.

We use the time stamps that result from sample alignment to initialize the Gaussian mixture[5]. Let $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T,N}$ consist of the realigned set of demonstrated trajectories. We initialize GMM with $K = T$ Gaussian functions. The parameters $\boldsymbol{\theta}^k$ (prior, mean, and covariance) of each Gaussian function are computed according to:

$$\boldsymbol{\theta}^k = \begin{cases} \pi^k = \frac{1}{K} \\ \boldsymbol{\mu}^k = mean\left(\boldsymbol{\Xi}^k\right) & \forall k \in 1..K-1 \\ \boldsymbol{\Sigma}^k = cov\left(\boldsymbol{\Xi}^k\right) + \sigma^0 \boldsymbol{I} \end{cases} \tag{4.21a}$$

$$\boldsymbol{\theta}^K = \begin{cases} \pi^K = \frac{1}{K} \\ \boldsymbol{\mu}^K \text{ is computed from Eq. (4.18a)} \\ \boldsymbol{\Sigma}^K = \sigma^0 \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I} \end{bmatrix} \end{cases} \tag{4.21b}$$

where $\boldsymbol{\Xi}^k$ denotes a subset of the demonstrations that belongs to the k-th Gaussian function, σ^0 is a small positive scalar to avoid numerical instability, and $\boldsymbol{I}$ is an identity matrix of the proper size. At initialization, $\boldsymbol{\Xi}^k$ is defined according to:

$$\boldsymbol{\Xi}^k = \{\xi^{k,n}, \dot{\xi}^{k,n}\}_{n=1}^{N} \tag{4.22}$$

We here assume that the obtained system after the initialization is stable. Practically, this is usually true as at initialization $K >> 1$, which results in having an accurate estimation of the motion in a region close to the demonstrations, and thus generating motions that reach the target in that region. When this is not true, one could resample the demonstration trajectories at a higher rate (e.g. through interpolation), and then redo the initialization step. The higher the sample rate, the more likely the system is stable after the initialization (remark that a stable system can be eventually obtained at the limit).

[5] The distortions resulting from aligning the trajectories are negligible if the sampling granularity is large.

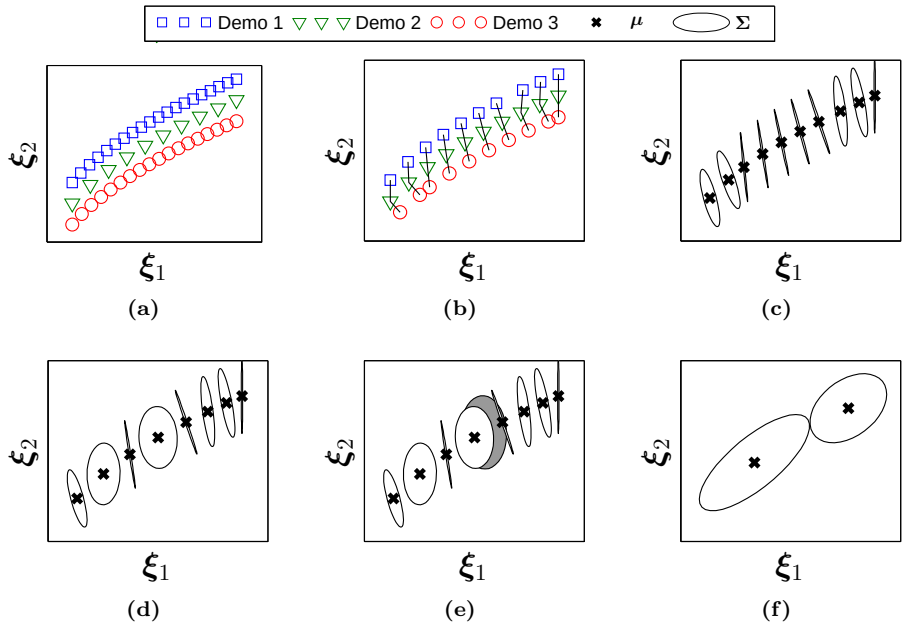


Figure 4.7: The Binary Merging (BM) learning algorithm. **(a)** Data points from three demonstrations. **(b)** Resultant trajectories after applying sample alignment. Points corresponding to the same time index are connected by a line. **(c)** Initialization step. **(d)** The GMM is updated by iteratively merging two pairs of adjacent Gaussians (2 with 3 and 5 with 6). **(e)** Because the new Gaussian resulting from the merging of 4 with 5 (shown in dark color) violates the stability criteria Eq. (4.18), the model remains unchanged. **(f)** Final model after termination.

Iteration: Iteration proceeds as follows. A pair $\{\boldsymbol{\theta}^k, \boldsymbol{\theta}^{k+1}\}$ of adjacent Gaussians is picked at random and merged into a new Gaussian function, by computing the new means and covariances on the union of data points associated to each of the two Gaussians. We will further denote the process of merging two Gaussians $\boldsymbol{\theta}^k$ and $\boldsymbol{\theta}^{k+1}$ with $\{\boldsymbol{\theta}^k + \boldsymbol{\theta}^{k+1}\}$. Then in the next step, both stability and accuracy conditions that are respectively given by Eqs. (4.18) and (4.20) are verified for the new model. If these conditions are satisfied, then the merged Gaussian function replaces the two selected Gaussian functions. The new model is now composed of $K-1$ Gaussians (see Fig. 4.7).

There are two key factors that should be taken into account during each iteration. Firstly, the stability condition given by Eq. (4.18a) directly imposes some constraints on the value of the mean and covariance matrix of the last Gaussian function. In fact, when the $K-1$ or K-th Gaussian function is selected, we directly update the value of $\boldsymbol{\mu}^K$ from Eq. (4.18a). As for the covariance matrix $\boldsymbol{\Sigma}^K$, when it does not satisfy this stability condition, we replace it with its closest covariance matrix that ensures Eq. (4.18a). Secondly, in contrast to the accuracy condition that is only computed for the demonstration datapoints, the stability conditions should be verified for all points in D. However in practice due to nonlinearities of $\boldsymbol{f}$, this can only be checked numerically on a mesh of datapoints defined over D. The required granularity of the mesh depends on the level of complexity of a motion. Using a low granular mesh for highly nonlinear

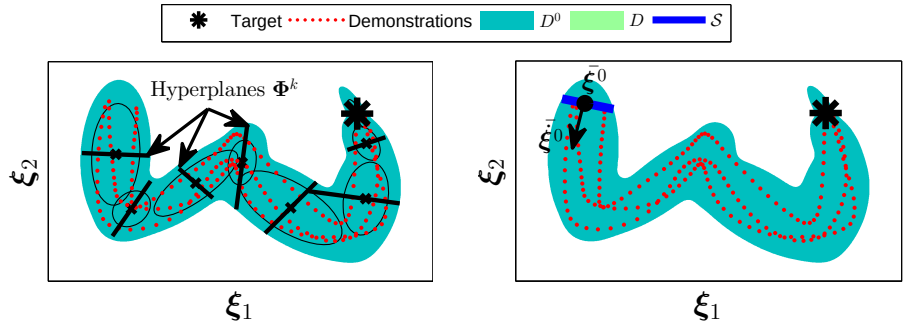
Algorithm 4.2 Checking stability conditions and defining D

Input: $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$, $\boldsymbol{\theta}$, and $\delta\kappa$
Compute $\delta = \min(\mathcal{P}(\boldsymbol{\xi}^{t,n}))$
Determine α and $D^0 = \{\, \boldsymbol{\xi} \in \mathbb{R}^d \;:\; \mathcal{P}(\boldsymbol{\xi}) \geq \alpha\delta \}$
Generate a uniform mesh $\mathcal{M}$ over D^0
if stability conditions of Theorem 4.1 on $\mathcal{M}$ are satisfied **then**
  Compute $\bar{\boldsymbol{\xi}}^0$ and $\bar{\dot{\boldsymbol{\xi}}}^0$
  Construct a hyper-plane $\boldsymbol{\Phi}$ containing $\bar{\boldsymbol{\xi}}^0$ and with normal $\bar{\dot{\boldsymbol{\xi}}}^0$.
  Compute the hyper-surface $\mathcal{S}$ from the intersection of D^0 and $\boldsymbol{\Phi}$
  Initialize the scale factor $\kappa \leftarrow 1$
  loop
    Generate trajectories $\forall \boldsymbol{\xi}^0 \in \mathcal{X}$ on the perimeter of $\mathcal{S}$
    if all reproduced trajectories are inside D^0 **then**
      Break
    else
      Decrease the scale factor by $\delta\kappa$, i.e. $\kappa \leftarrow \kappa - \delta\kappa$
      Scale down $\mathcal{S}$ with the scale factor κ
    end if
  end loop
  Define D to be a region confined by the reproduced trajectories.
  return true & D
else
  return false
end if

function may result in error in evaluating the stability of $\boldsymbol{f}$. Thus it is necessary to tune the granularity of a mesh such that it captures the nonlinearity of the dynamics while keeping the computation time as small as possible.

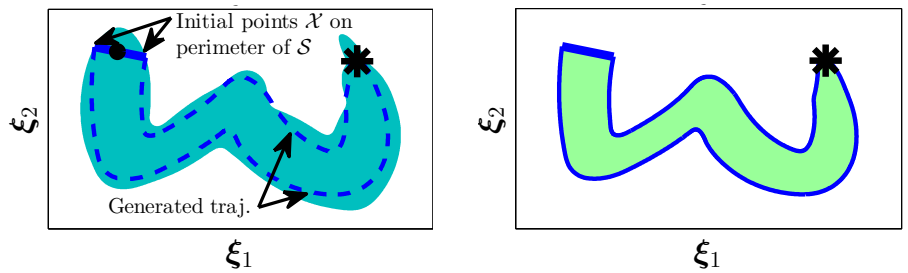
Algorithm 4.2 illustrates with pseudocode how to verify the stability conditions. The main inputs to the algorithm are the demonstrations and the model parameters. At the first step, the algorithm computes the positive scalar δ over the dataset. Then an initial rough estimation of the domain D^0 is computed based on the value of δ (see Fig. 4.8a). At this stage, the resulting domain D^0 is not necessarily stable. In the next step, the stability conditions of Theorem 4.1 are verified over a mesh $\mathcal{M}$ on D^0. If the system satisfies these conditions, then the algorithm computes an invariant set D inside D^0 as follows: First, it constructs a hyper-plane $\boldsymbol{\Phi}$ with a point $\bar{\boldsymbol{\xi}}^0 = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{\xi}^{0,n}$ (the mean of the position of all demonstrations at time $t = 0$) and the normal vector $\bar{\dot{\boldsymbol{\xi}}}^0 = \frac{1}{N}\sum_{n=1}^{N} \dot{\boldsymbol{\xi}}^{0,n}$ (the mean of the velocity of all demonstrations at $t = 0$). The intersection of $\boldsymbol{\Phi}$ and D^0 forms a hyper-surface $\mathcal{S}$ (see Fig. 4.8b). The algorithm then generates trajectories for all starting points $\boldsymbol{\xi}^0 \in \mathcal{X}$ on the perimeter of $\mathcal{S}$ (see Fig. 4.8c). If all generated trajectories remain inside D^0, then we consider the resulting hyper-tube inclosed by trajectories as D (see Fig. 4.8d). Otherwise, the hyper-surface $\mathcal{S}$ is scaled down slightly and the procedure repeated again. Figs. 4.8e and 4.8f show a 3D illustration of the process.

The BM learning algorithm terminates when it is no longer possible to merge any pair of Gaussian functions without violating the maximum accepted error or becoming unstable. The model converges within a maximum $T(T-1)/2$



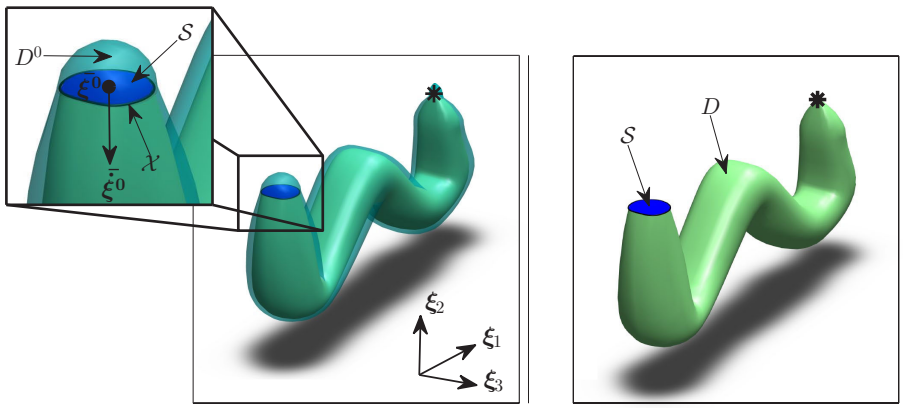
(a) Computing a rough estimation of the stability domain D^0.

(b) Generating the hyper-surface $\mathcal{S}$ using the point $\bar{\boldsymbol{\xi}}^0$ and the normal vector $\dot{\bar{\boldsymbol{\xi}}}^0$.



(c) Generating boundary trajectories starting from all the points $\boldsymbol{\xi}^0 \in \mathcal{X}$ on the perimeter of $\mathcal{S}$.

(d) If all boundary trajectories remain inside D^0, we use them to define D.



(e) 3D representation of the domains D^0, the hyper-surface $\mathcal{S}$, and the set of initial points $\mathcal{X}$.

(f) 3D representation of the domains D.

Figure 4.8: Finding the stability domain D. Starting from any point inside D, the trajectory converges asymptotically to the target.

iterations[6]. Such a learning procedure results in a higher number of Gaussian functions along curvatures in the motion (e.g. observe that the straight parts in Fig. 4.6 require fewer Gaussian functions than the highly curved parts). Note that BM does not ensure that the globally minimal number of Gaussian functions is obtained (the algorithm finds a local minimum), and several derivations of the model may be done to discover via random sampling a better solution than the initial one.

4.4.3 Experimental Results

The presented algorithm is validated to control the point-to-point motions of a 6-DoF Katana-T arm and the 4-DoF right arm of the humanoid robot Hoap-3. Both robots are equipped with a built-in PID controller to follow a reference motion. Training data was provided by a human expert 3 to 5 times for each example by back-driving the robot. The first task, represented in Fig. 4.9, consists of having the Hoap-3 robot draw lines in a constrained 2D area. The second task requires the Katana-T arm to put an object into a container while avoiding an obstacle (see Fig. 4.10). The position of the container and the obstacle are detected from an external stereo-vision system.

We use Cartesian coordinates system to represent the motion (the axes $\boldsymbol{\xi}_1$, $\boldsymbol{\xi}_2$, and $\boldsymbol{\xi}_3$ correspond respectively to x, y, and z in the Cartesian coordinates system). These tasks illustrate well the importance of having a locally stable controller that closely follows the learned dynamics. Observe that BM implicitly infers the constraints of the motion from demonstrations by accurately following demonstrations in the stability domain D. In both experiments, the task is shown to the robot three times. BM learning procedure results in locally asymptotically stable models with $K = 9$ and $K = 6$ for the first and the second experiments, respectively. The learned model are then used to generate motions within the stability domain. As can be seen in Figs. 4.9 and 4.10, the robot is able to successfully reproduce similar motions to demonstrations within the stability region drawn in green.

One of the main advantages of DS is its instant adaptation to external perturbations. Figure 4.11 clearly illustrates such a result for the two mentioned experiments, where the robots' end-effector are displaced twice during the tasks execution. Right after applying perturbations, our model is able to recompute a new trajectory based on the current position of the robot's end-effector. Note that since the models constructed by BM are only locally asymptotically stable, the robustness of the model is only ensured if perturbations do not send trajectories (i.e. robot's end-effector) outside D.

[6] At each iteration, there are $K - 1$ possible combinations of adjacent Gaussian functions. The maximum number of iterations happens in the most unlucky case where from the initialization to the last iteration, only the model from the merging of the last possible pair of Gaussian functions satisfies the stability and accuracy criteria. Thus, at each iteration, the algorithm tries all $K - 1$ combinations, which yields the upper bound $i_{max} = \sum_{k=2}^{T}(k-1) = T(T-1)/2$ for the maximum number of iterations.

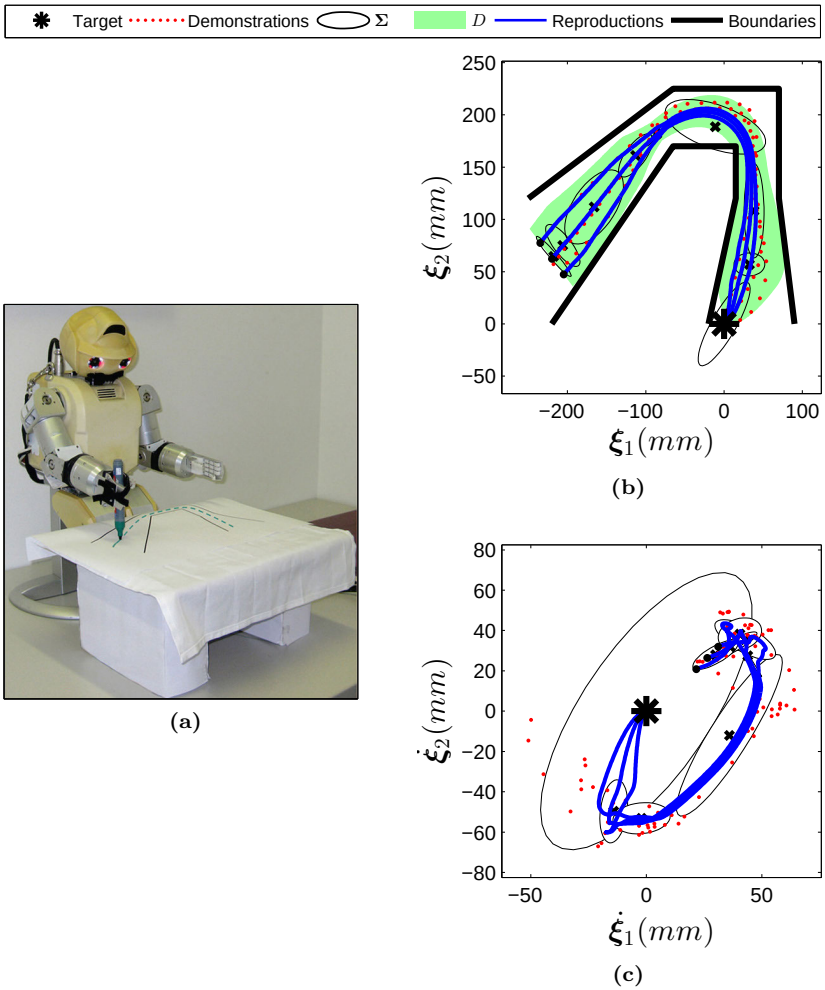


Figure 4.9: The Hoap-3 robot performing the experiment of drawing lines in a constrained 2D area.

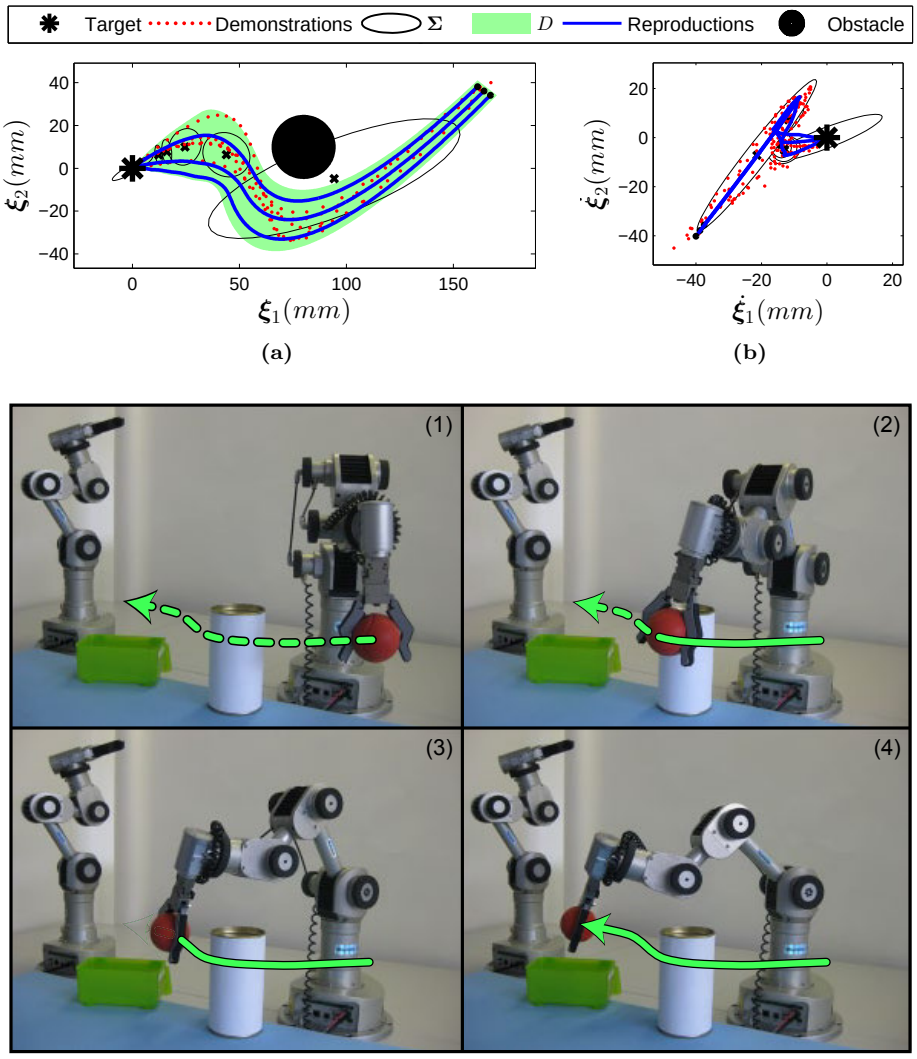


Figure 4.10: The Katana-T arm performing the experiment of moving an object while avoiding an obstacle.

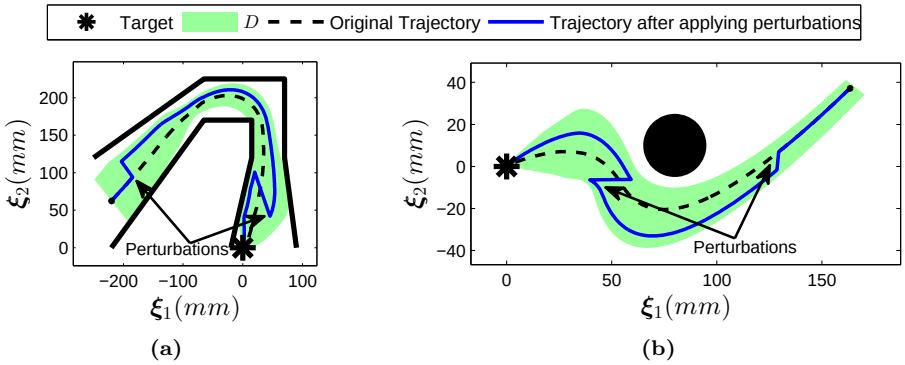


Figure 4.11: On-line adaptation of motion to abrupt spatial perturbations.

4.4.4 Discussion and Conclusion

In this section, we presented the Binary Merging (BM) learning method for encoding point-to-point motions with first order autonomous nonlinear ODE. BM offers a framework to build a locally stable estimate of nonlinear DS from a set of demonstrations. The estimated DS generates trajectories that accurately follow the motion dynamics based on the metric of accuracy defined by the user. More importantly, the DS model of the motion is inherently robust to perturbations within the stability domain D, which removes the need for a heuristic to regenerate a new trajectory in the face of perturbations.

There are, however, a number of shortcomings when using BM, which can be sorted in the order of their importance as follows:

1. As it can be seen in Figs. 4.9 and 4.10, the stability domain D usually corresponds to a narrow region around the demonstrations, which could be quite limiting for tasks that require a large domain of applicability.

2. At initialization, BM uses sample alignment which is very sensitive to demonstrations, and by construction is only effective when demonstrations are very similar (in terms of the temporal order, position, and velocity). This is more limiting than our original assumption on the consistency of demonstrations (see Section 4.1). For example, consider using BM to learn a 2D motion based on two demonstration trajectories, that start from two different initial points placed on the positive and negative directions of the x-axis. Although these two demonstrations do not contradict each other, the sample alignment algorithm would fail as there is no similarity between their speed profile.

3. BM relies on determining numerically the stability region. Its computational cost grows exponentially with d, polynomially with the granularity of the mesh, and quadratically with T (the number of Gaussian functions at initialization). Hence, it could already become prohibitive in 3D and intractable in higher dimensions.

4. As the stability conditions are only verified on a mesh over the region D, the validity of the result strongly depends on the quality of the meshing. The finer the granularity of the mesh, the more likely the stability is guaranteed in D. However, as outlined before, increasing the granularity of the mesh is costly and a tradeoff must hence be found between increasing likelihood of a good coverage and limiting computational costs.

These drawbacks may be very limiting for many robot experiments. In the next section we present an alternative learning algorithm that can overcome the above shortcomings.

4.5 Stable Estimator of Dynamical Systems

In this section we define sufficient conditions to ensure global asymptotic stability of our estimate of a nonlinear autonomous DS at the target. We then propose a learning method, called *Stable Estimator of Dynamical Systems (SEDS)*, to learn the parameters of the DS so as to ensure all motions follow closely the demonstrations while ultimately reaching and stopping at the target. Being time-invariant and globally asymptotically stable at the target, a DS that is estimated with SEDS can respond immediately and appropriately to perturbations encountered during the motion. We evaluate this method through a set of robot experiments and on a library of human handwriting motions.

This section is structured as follows. In Section 4.5.1 we first develop conditions for ensuring global asymptotic stability of nonlinear DS. Then, we propose a learning method to build an estimate of nonlinear DS subject to these conditions. In Section 4.5.2, we quantify the performance of the proposed method in simulation and robot experiments. We devote Section 4.5.3 to discussion and conclusion.

4.5.1 Learning Globally Stable Models

In this section we provide a set of stability conditions to ensure global asymptotic stability of $\boldsymbol{f}$ at the target. Similarly to our previous approach, we use GMM to encode the dynamics of the motions. However, as opposed to BM, here we take into account the effect of all Gaussian functions without any need to truncate the estimate to solely using the adjacent Gaussian functions.

Our approach is based on the following fundamental physical observation: "if the total energy of a mechanical system is continuously dissipated, then the system, whether linear or nonlinear, must eventually settle down to an equilibrium point" (Slotine & Li, 1991). Thus in order to reach our goal in building a globally asymptotically stable DS, we need to set the parameters of GMR so that by starting the motion from any point in the state space, the energy of the system decreases as the motion evolves until it reaches the target, where it becomes zero. The latter can be achieved by ensuring the following stability conditions.

□

Theorem 4.2 *Assume that the state trajectory evolves according to Eq. (4.8). Then the function described by Eq. (4.8) is globally asymptotically stable at its unique equilibrium point* $\boldsymbol{\xi}^*$ *in* $\mathbb{R}^d$ *if:*

$$\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}} = \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1}(\boldsymbol{\mu}^k_{\boldsymbol{\xi}} - \boldsymbol{\xi}^*) \qquad \forall k = 1..K \tag{4.23a}$$

$$\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} + (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1}(\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}})^T \prec 0 \qquad \forall k = 1..K \tag{4.23b}$$

where $. \prec 0$ *refers to the negative definiteness of a matrix.*

Proof: See Appendix A.2.

■

The conditions given by Eq. (4.23) impose constraints on each Gaussian function so that the energy dissipation due the presence of that Gaussian becomes negative everywhere except at the target, where it becomes zero. As the final estimate from GMR is determined by summing (with a set of positive nonlinear weights) the local estimates from Gaussian functions, the total energy dissipation of the system is thus by construction negative everywhere except at the target, where it becomes zero.

Theorem 4.2 provides us with sufficient conditions whereby the estimate $\boldsymbol{f}(\boldsymbol{\xi})$ is globally asymptotically stable at the target. It remains now to determine a procedure for computing the unknown parameters of Eq. (4.8), i.e. $\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}^1..\boldsymbol{\mu}^K; \boldsymbol{\Sigma}^1..\boldsymbol{\Sigma}^K\}$ while satisfying these stability conditions. In this section we propose a learning algorithm, called *Stable Estimator of Dynamical Systems (SEDS)*, that computes optimal values of $\boldsymbol{\theta}$ by solving an optimization problem under the constraint of ensuring the model's global asymptotic stability at the target. We consider two different candidates for the optimization objective function: 1) log-likelihood, and 2) Mean Square Error (MSE). We will evaluate and compare the results from both approaches in Section 4.5.2.1.

SEDS-Likelihood: *Using log-likelihood as a means to quantify the accuracy of estimations based on demonstrations.*

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \log \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | \boldsymbol{\theta}) \tag{4.24}$$

subject to

$$\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}} = \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} (\boldsymbol{\mu}^k_{\boldsymbol{\xi}} - \boldsymbol{\xi}^*) \qquad \forall k \in 1..K \tag{4.25a}$$

$$\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} + (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} (\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}})^T \prec 0 \qquad \forall k \in 1..K \tag{4.25b}$$

$$\boldsymbol{\Sigma}^k \succ 0 \qquad \forall k \in 1..K \tag{4.25c}$$

$$0 < \pi^k \leq 1 \qquad \forall k \in 1..K \tag{4.25d}$$

$$\sum_{k=1}^{K} \pi^k = 1 \tag{4.25e}$$

where $\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | \boldsymbol{\theta})$ *is given by Eq. (4.6), and* $\mathcal{T} = \sum_{n=1}^{N} T^n$ *is the total number of training data points.*

The first two constraints in Eq. (4.25) are stability conditions from Theorem 4.2. The last three constraints are imposed by the nature of the Gaussian Mixture Model to ensure that $\boldsymbol{\Sigma}^k$ are positive definite matrices, priors π^k are positive scalars smaller than or equal to one, and the sum of all priors is equal to one (because the probability value of Eq. (4.6) should not exceed one).

SEDS-MSE: *Using Mean Square Error as a means to quantify the accuracy of estimations based on demonstrations.*

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \left(\boldsymbol{f}(\boldsymbol{\xi}^{t,n}) - \dot{\boldsymbol{\xi}}^{t,n}\right)^T \left(\boldsymbol{f}(\boldsymbol{\xi}^{t,n}) - \dot{\boldsymbol{\xi}}^{t,n}\right) \tag{4.26}$$

subject to the same constrains as given by Eq. (4.25). In Eq. (4.26), $\boldsymbol{f}(\boldsymbol{\xi}^{t,n})$ are computed directly from Eq. (4.8).

Both SEDS-Likelihood and SEDS-MSE corresponds to a constrained optimization problem that can be solved using the techniques presented in Section 2.3. In order to improve the optimization performance, we use an analytic expression of the required optimization derivatives. Furthermore, we suggest a reformulation to the above optimization problems that guarantees the optimization constraints given by Eqs. (4.25c) to (4.25e) are satisfied. The analytical formulation of derivatives and the mathematical reformulation to satisfy the optimization constraints are explained in details in Appendix C.

Note that for both of the above optimization problems, a feasible solution always exists. Algorithm 4.3 provides a simple and efficient way to compute a feasible initial guess for the optimization parameters. Starting from an initial value, the solver tries to optimize the value of $\boldsymbol{\theta}$ such that the cost function J is minimized. However since the proposed optimization problems are non-convex, one cannot ensure to find the globally optimal solution. Solvers are usually very sensitive to initialization of the parameters and will often converge to some local minima of the objective function. Based on our experiments, running the optimization with the initial guess obtained from Algorithm 4.3 usually results in a good local minimum. In all experiments reported in Section 4.5.2, we ran the initialization three to four times, and use the result from the best run for the performance analysis.

For the SEDS-Likelihood approach, we use the Bayesian Information Criterion (BIC) to choose the optimal number of Gaussian functions K. BIC determines a tradeoff between optimizing the model's likelihood and the number of parameters needed to encode the data:

$$BIC = \mathcal{T} J(\boldsymbol{\theta}) + \frac{n_p}{2} \log(\mathcal{T}) \tag{4.27}$$

where $J(\boldsymbol{\theta})$ is the normalized log-likelihood of the model given by Eq. (4.24), and n_p is the total number of free parameters.

For the SEDS-MSE approach, however, it is not possible to use the BIC as this approach does not optimize for maximizing the likelihood. To obtain an accurate model without overfitting the demonstrations, one can split the demonstrations into training and test datasets. The optimal number of Gaussian functions corresponds to the minimum value of K that provides an accurate estimate on both datasets.

Algorithm 4.3 Procedure to determine an initial guess for the optimization parameters

Input: $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$ and K

1: Run EM over demonstrations to find an estimate of π^k, $\boldsymbol{\mu}^k$, and $\boldsymbol{\Sigma}^k$, $k \in 1..K$.
2: Define $\tilde{\pi}^k = \pi^k$ and $\tilde{\boldsymbol{\mu}}^k_{\boldsymbol{\xi}} = \boldsymbol{\mu}^k_{\boldsymbol{\xi}}$
3: Transform covariance matrices such that they satisfy the optimization constraints given by Eqs. (4.25b) and (4.25c):

$$\begin{cases} \tilde{\boldsymbol{\Sigma}}^k_{\boldsymbol{\xi}} = \boldsymbol{I} \circ abs(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}}) \\ \tilde{\boldsymbol{\Sigma}}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} = -\boldsymbol{I} \circ abs(\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}) \\ \tilde{\boldsymbol{\Sigma}}^k_{\dot{\boldsymbol{\xi}}} = \boldsymbol{I} \circ abs(\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}}) \\ \tilde{\boldsymbol{\Sigma}}^k_{\boldsymbol{\xi}\dot{\boldsymbol{\xi}}} = -\boldsymbol{I} \circ abs(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}\dot{\boldsymbol{\xi}}}) \end{cases} \qquad \forall k \in 1..K$$

where $\circ$ and $abs(.)$ corresponds to entrywise product and absolute value function, and $\boldsymbol{I}$ is a $d \times d$ identity matrix.
4: Compute $\tilde{\boldsymbol{\mu}}^k_{\dot{\boldsymbol{\xi}}}$ by solving the optimization constraint given by Eq. (4.25a):

$$\tilde{\boldsymbol{\mu}}^k_{\dot{\boldsymbol{\xi}}} = \tilde{\boldsymbol{\Sigma}}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} (\tilde{\boldsymbol{\Sigma}}^k_{\boldsymbol{\xi}})^{-1} (\tilde{\boldsymbol{\mu}}^k_{\boldsymbol{\xi}} - \boldsymbol{\xi}^*)$$

Output: $\boldsymbol{\theta}^0 = \{\tilde{\pi}^1..\tilde{\pi}^K; \tilde{\boldsymbol{\mu}}^1..\tilde{\boldsymbol{\mu}}^K; \tilde{\boldsymbol{\Sigma}}^1..\tilde{\boldsymbol{\Sigma}}^K\}$

The SEDS-Likelihood approach requires the estimation of $K(1 + 3d + 2d^2)$ parameters (the priors π^k, the means $\boldsymbol{\mu}^k$, and the covariance matrices $\boldsymbol{\Sigma}^k$ are of size 1, $2d$ and $d(2d + 1)$ respectively). However, the number of parameters can be reduced since the constraints given by Eq. (4.25a) provide an explicit formulation to compute $\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}$ from other parameters (i.e. $\boldsymbol{\mu}^k_{\boldsymbol{\xi}}$, $\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}}$, and $\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}$). Thus the total number of parameters to construct a GMM with K Gaussian functions is $K(1+2d(d+1))$. As for SEDS-MSE, the number of parameters is even more reduced since when constructing $\boldsymbol{f}$, the term $\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}}$ is not used and thus can be omitted during the optimization. Taking this into account, the total number of learning parameters for the SEDS-MSE reduces to $K(1 + \frac{3}{2}d(d+1))$.

For both approaches, learning grows linearly with the number of Gaussian functions and quadratically with the dimension. In comparison, the number of parameters in the proposed method is smaller than those needed for GMM[7]. The retrieval time of both approaches is short and in the same order of GMR.

4.5.2 Experimental Evaluations

In this section we evaluate the performance of SEDS in a series of simulation and robot experiments. Precisely, in Section 4.5.2.1 we compare the performance of the SEDS method against a library of 20 human handwriting motions when using either the likelihood or MSE. In Section 4.5.2.2, we validate SEDS to estimate the DS motion of two robots (a) the 7-DoF right arm of the humanoid robot iCub, and (b) the 6-DoF industrial robot Katana-T arm. In Section 4.5.2.3, we show how SEDS can be used to learn second or higher order dynamics, and finally in Section 4.5.2.4 we demonstrate through an example the possibility to embed different local behaviors in a single DS.

[7]The number of learning parameter in GMR is $K(1 + 3d + 2d^2)$.

4.5.2.1 SEDS TRAINING: LIKELIHOOD VS. MSE

In Section 4.5.1 we proposed two objective functions: likelihood and MSE for training DS models. We compare the results obtained with each method for modeling 20 handwriting motions. The demonstrations are collected from pen input using a Tablet-PC. Figure 4.12 shows a qualitative comparison of the estimate of handwriting motions. All reproductions were generated in simulation to exclude the error due to the robot controller from the modeling error. The accuracy of the estimate is measured by computing the so-called "swept error area":

$$\mathscr{E} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T^n}\mathcal{A}(\boldsymbol{\xi}^n(t), \boldsymbol{\xi}^n(t+1), \boldsymbol{\xi}^{t,n}, \boldsymbol{\xi}^{t+1,n}) \tag{4.28}$$

where $\mathcal{A}(\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3, \boldsymbol{p}^4)$ corresponds to the area of the tetragon generated by the four points $\boldsymbol{p}^1$ to $\boldsymbol{p}^4$, $\boldsymbol{\xi}^n(t) = \sum_{i=0}^{t}\dot{\boldsymbol{\xi}}^n(i)dt$ generate an estimate of the corresponding demonstrated trajectory $\boldsymbol{\xi}^n$ by starting from the same initial points as those demonstrated, i.e. $\boldsymbol{\xi}^n(0) = \boldsymbol{\xi}^{0,n}$, $\forall n \in 1..N$. Figure 4.13 shows an example of the swept error area between the reference and the generated trajectory for a 2D motion.

The quantitative comparison between the two methods is represented in Fig. 4.14. SEDS-Likelihood slightly outperforms SEDS-MSE in accuracy of the estimate. This could be due to the fact that Eq. (4.26) only considers the norm of $\dot{\boldsymbol{\xi}}$ during the optimization, while in computing the swept error area the direction of $\dot{\boldsymbol{\xi}}$ is also important (see Eq. (4.28)). Though one could improve the performance of SEDS-MSE by considering the direction of $\dot{\boldsymbol{\xi}}$ in Eq. (4.26), this would make the optimization problem more difficult to solve by changing a convex objective function into a non-convex one[8].

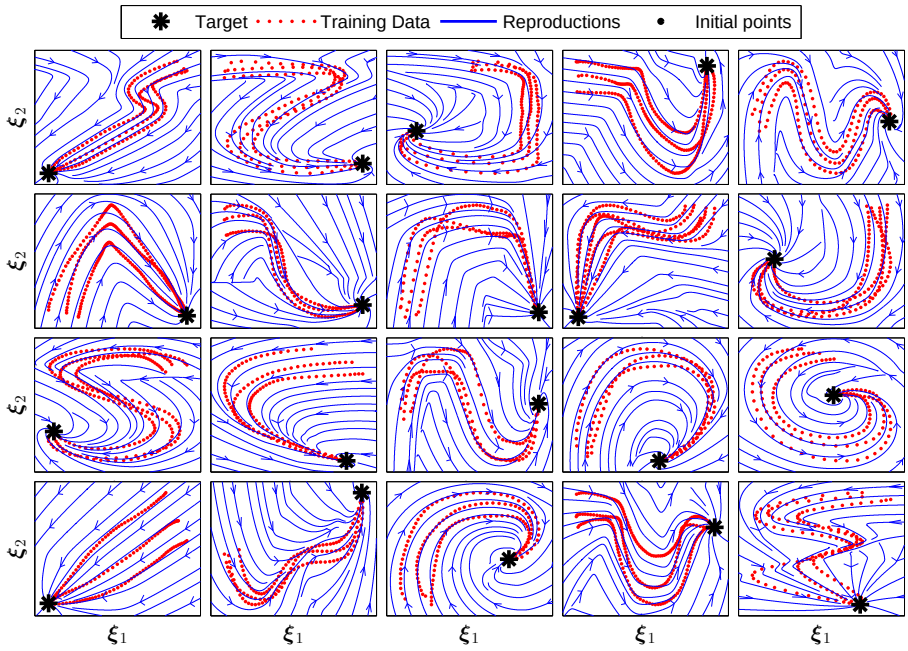
SEDS-MSE is advantageous over SEDS-Likelihood in that it requires fewer parameters (this number is reduced by a factor of $\frac{1}{2}Kd(d+1)$). Furthermore, SEDS-MSE training is lower than SEDS-Likelihood as it deals with a simpler optimization problem. Following the above observations, we could clearly see that the difference between the two approaches are very small. For brevity, we will choose SEDS-Likelihood in the rest of experiments as it results to a slightly more accurate model[9].

[8] Alternatively, one can use a different cost function that propagates the effect of the estimation error at each time step along the generated trajectory (as opposed to the MSE cost function given by Eq. (4.26) that assumes independency across datapoints):

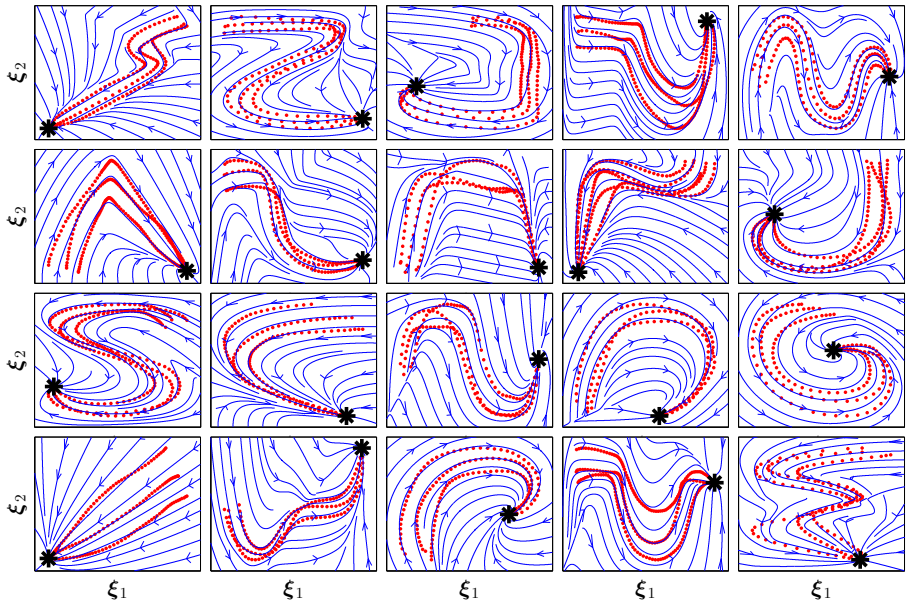
$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T^n}\left(\omega_{\boldsymbol{\xi}}\|\boldsymbol{\xi}^n(t) - \boldsymbol{\xi}^{t,n}\|^2 + \omega_{\dot{\boldsymbol{\xi}}}\|\dot{\boldsymbol{\xi}}^n(t) - \dot{\boldsymbol{\xi}}^{t,n}\|^2\right)$$

$\dot{\boldsymbol{\xi}}^n(t) = \boldsymbol{f}(\boldsymbol{\xi}^n(t))$ are computed directly from Eq. (4.8), and $\omega_{\boldsymbol{\xi}}$ and $\omega_{\dot{\boldsymbol{\xi}}}$ are positive scalars weighing the influence of the position and velocity terms in the cost function. The above MSE cost function could result in a more accurate estimation of $\boldsymbol{f}$ as it has a better metric of evaluation. However, it does that at the cost of solving a much more complex optimization problem.

[9] Note that in our experiments the differences between the two algorithms in terms of the number of parameters and the training time are small. As the training is done offline, they are not decisive factors.



(a) SEDS-Likelihood



(b) SEDS-MSE

Figure 4.12: Performance comparison of SEDS-Likelihood and SEDS-MSE against a library of 20 human handwriting motions.

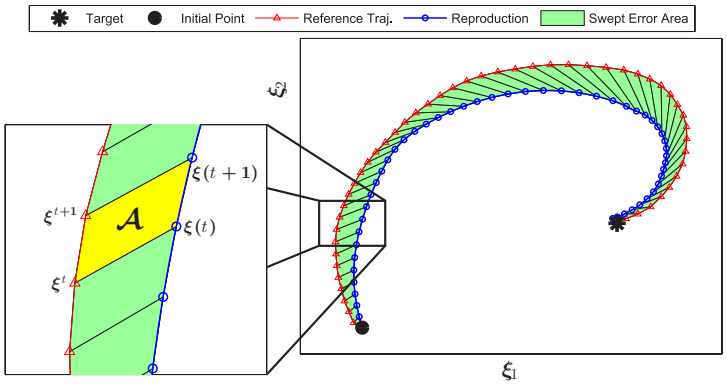


Figure 4.13: Illustration of the swept error area between the reference and the generated trajectories.

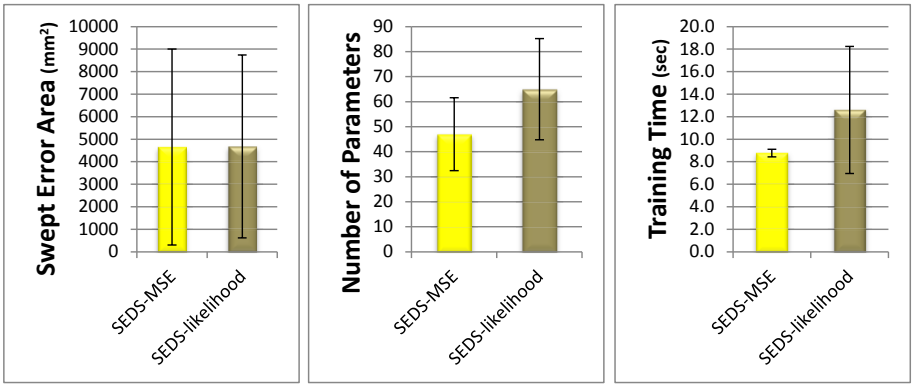


Figure 4.14: Performance comparison of SEDS-Likelihood and SEDS-MSE through a library of 20 human handwriting motions.

4.5.2.2 Learning Discrete Motions in the Operational Space

We report on five robot experiments to teach the Katana-T and the iCub robot to perform nonlinear point-to-point motions. Both robots are kinematically driven and have a built-in PID controller. In all our experiments the origin of the reference coordinates system is attached to the target. The motion is hence controlled with respect to this frame of reference. Such representation makes the parameters of a DS invariant to changes in the target position. The target position is detected at the rate of 50 fps using two high-speed Mikrotron MK-1311 cameras.

In the first experiment, we teach the 6-DoF industrial Katana-T arm how to put small blocks into a container[10] (see Fig. 4.15). We use the Cartesian coordinates system to represent the motions. In order to have human-like motions, the learned model should be able to generate trajectories with both similar position and velocity profiles to the demonstrations. In this experiment, the task was shown to the robot six times, and was learned using $K = 6$ Gaussian functions.

[10]The robot is only taught how to move blocks. The problem of grasping the blocks is out of the scope of this thesis. Throughout the experiments, we pose the blocks such that they can be easily grasped by the robot.

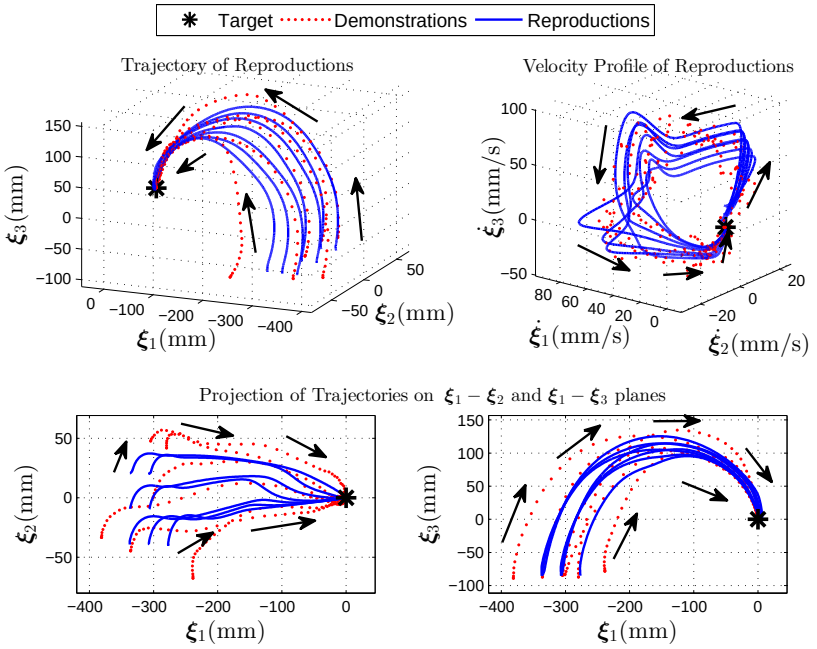
Figure 4.15a illustrates the obtained results for generated trajectories starting from different points in the task space. The direction of motion is indicated by arrows. All reproduced trajectories are able to follow the same dynamics (i.e. having similar position and velocity profile) as the demonstrations.

Immediate Adaptation: Fig. 4.15b shows the robustness of the model to the change in the environment. In this graph, the original trajectory is plotted in thin blue line. The thick black line represents the generated trajectory for the case where the target is displaced at $t = 1.5$ second. Having defined the motion as an autonomous DS, the adaptation to the new target's position can be done instantly.

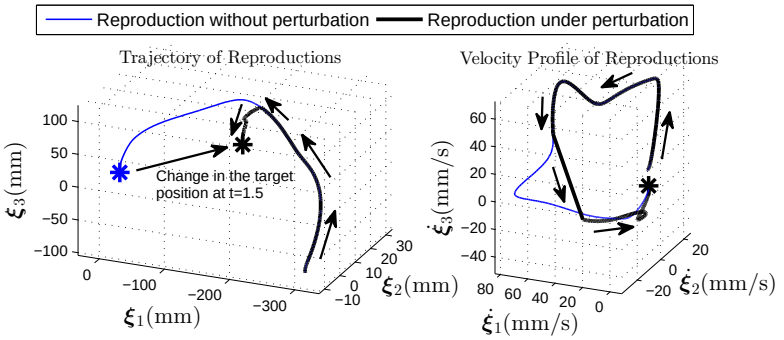
Increasing Accuracy of Generalization: While convergence to the target is always ensured from conditions given by Eq. (4.23), due to the lack of information for points far from demonstrations, the model may reproduce some trajectories that are not consistent with the usual way of doing the task. For example, consider Fig. 4.16a, i.e. when the robot starts the motion from the left-side of the target, it first turns around the container and then approaches the target from its right-side. This behavior may not be optimal as one expects the robot to follow the shortest path to the target and to reach it from the same side as the one it started from. However, such a result is inevitable since the information given by the teacher is incomplete, and thus the inference for points far from the demonstrations is not reliable. In order to improve the task execution, it is necessary to provide the robot with more demonstrations (information) over regions not covered before. By showing the robot more demonstrations and re-training the model with the new data, the robot is able to successfully accomplish the task (see Fig. 4.16b).

The second and third experiments consisted of having the Katana-T robot place a saucer at the center of the tray and putting a cup on the top of the saucer. Both tasks were shown 4 times and were learned using $K = 4$ Gaussians. The experiments and the generalization of the tasks starting from different points in the space are shown in Figs. 4.17 and 4.18. The adaptation of both models in the face of perturbations are also illustrated in Fig. 4.19. Note that in this experiment the cup task is executed after finishing the saucer task; however, for convenience we superimpose both tasks in the same graph. In both tasks the target (the saucer for the cup task and the tray for the saucer task) is displaced during the execution of the task at $t = 2$ seconds. In both experiments, the adaptation to the perturbation is handled successfully.

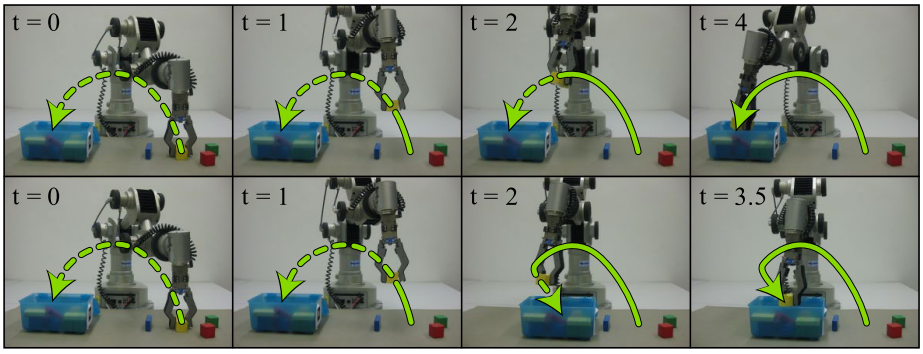
The forth and fifth experiments consisted of having the 7-DoF right arm of the humanoid robot iCub perform complex motions, containing several nonlinearities (i.e. successive curvatures) in both position and velocity profiles. Similar to above, we use the Cartesian coordinates system to represent these motions. The tasks are shown to the robot by teleoperating it using motion sensors (see



(a) The ability of the model to reproduce similar trajectories starting from different points in the space.

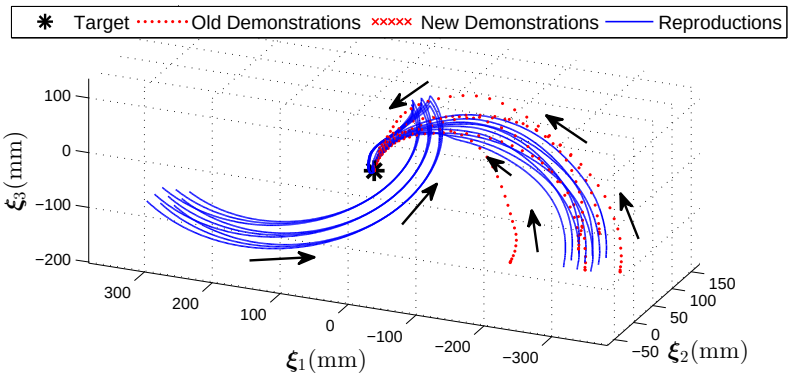


(b) The ability of the model to adapt its trajectory on-the-fly to a change in the target's position

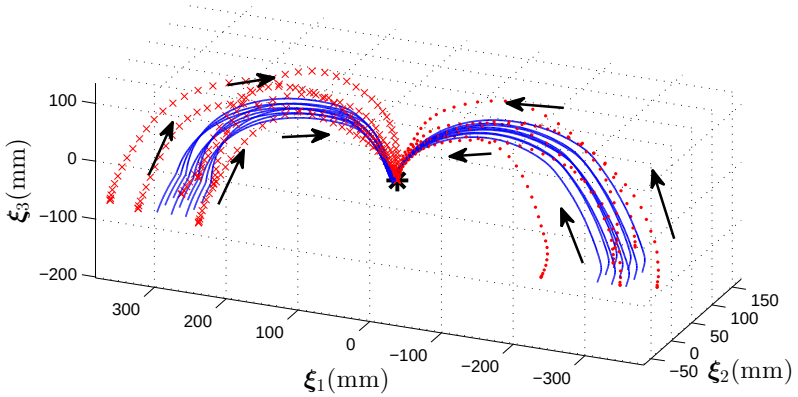


(c) Stills showing the reproduction of the task by the robot in the absence and presence of perturbation.

Figure 4.15: The Katana-T arm performing the experiment of putting small blocks into a container. Please see the text for further information.



(a) Generalization based on the original model.



(b) Generalization after retraining the model with the new and old datasets.

Figure 4.16: Improving the task execution by adding more data for regions far from the demonstrations.

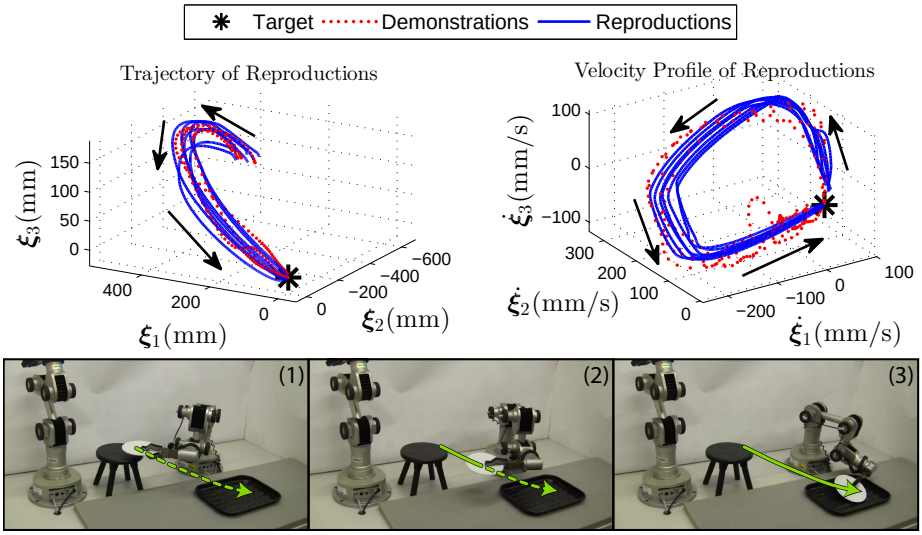


Figure 4.17: The Katana-T arm performing the experiment of putting a saucer on a tray.

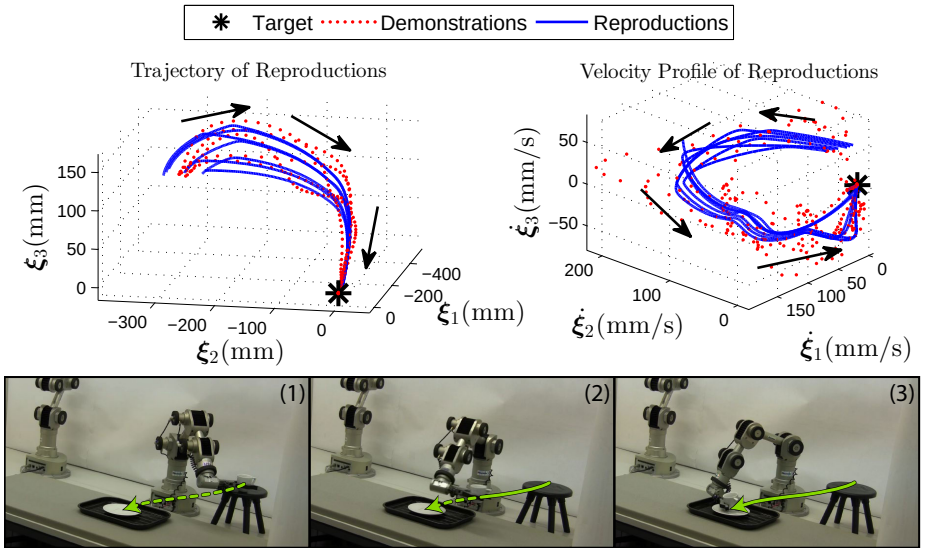


Figure 4.18: The Katana-T arm performing the experiment of putting a cup on a saucer.

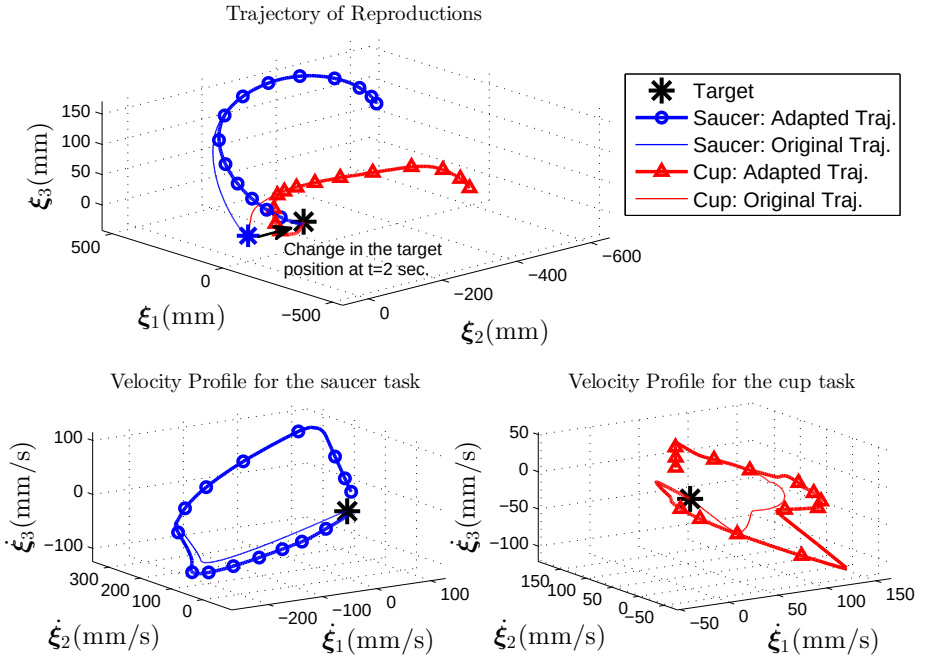


Figure 4.19: The ability of the model to instantly adapt its trajectory to a change in the target's position.

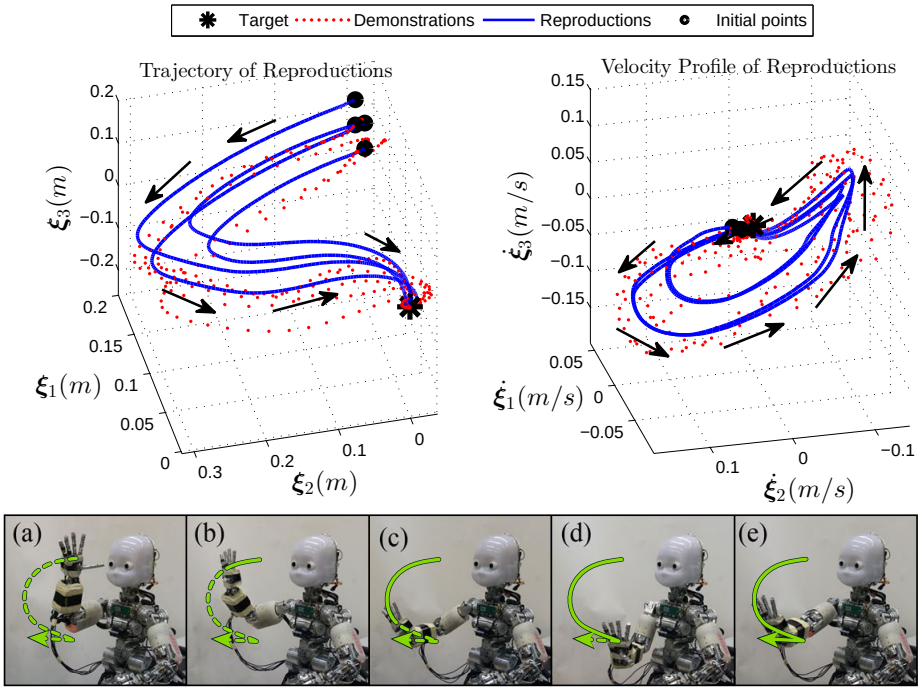


Figure 4.20: The first experiment with the iCub. The robot does a semi-spiral motion toward its right-side, and at the bottom of the spiral, it stretches forward its hand completely.

Fig. 4.1). Figure 4.20 illustrates the result for the first task where the iCub starts the motion in front of its face. Then it does a semi-spiral motion toward its right-side, and finally at the bottom of the spiral, it stretches forward its hand completely. In the second task, the iCub starts the motion close to its left fore-hand. Then it does a semi-circle motion upward and finally brings its arm completely down (see Fig. 4.21). The two experiments were learned using 5 and 4 Gaussian functions, respectively. In both experiments the robot is able to successfully follow the demonstrations and to generalize the motion for several trajectories with different starting points. Similarly to what was observed in the three experiments with the Katana-T robot, the models obtained for the iCub's experiments are robust to perturbations.

4.5.2.3 Learning Second Order Dynamics

So far we have shown how DS can be used to model/learn a demonstrated motion when modeled as a first order time-invariant ODE. Though this class of ODE functions are generic enough to represent a wide variety of robot motions, they fail to accurately define motions that rely on second order dynamics such as a self-intersecting trajectory or motions for which the starting and final points coincide with each other (e.g. a triangular motion). Critical to such kinds of motions is the ambiguity in the correct direction of velocity at the intersection point if the model's variable $\boldsymbol{\xi}$ considered to be only the cartesian position (i.e. $\boldsymbol{\xi} = \boldsymbol{x} \Rightarrow \dot{\boldsymbol{\xi}} = \dot{\boldsymbol{x}}$). This ambiguity usually results in skipping the loop part of

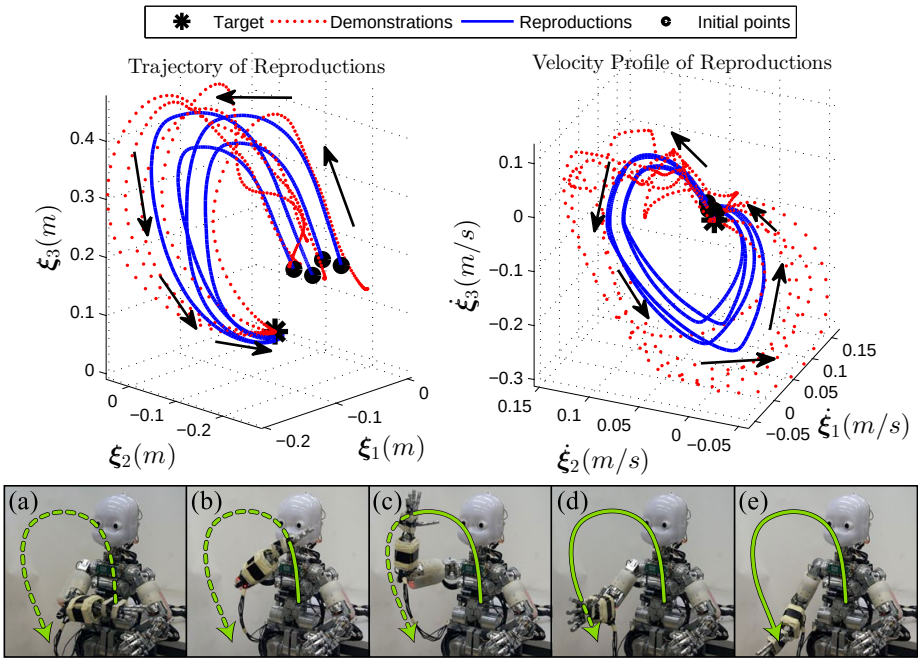


Figure 4.21: The second experiment with the iCub. The robot does a semi-circle motion upward and then brings its arm completely down.

the motion. However, in this example, this problem can be solved if one defines the motion in terms of position, velocity, and acceleration, i.e. a second order dynamics:

$$\ddot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \tag{4.29}$$

where $\boldsymbol{g}$ is an arbitrary function. Observe that any second order dynamics in the form of Eq. (4.29) can be easily transformed into a first-order ODE through a change of variable, i.e.:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{v} \\ \dot{\boldsymbol{v}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}) \end{cases} \Rightarrow [\dot{\boldsymbol{x}}; \dot{\boldsymbol{v}}] = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{v}) \tag{4.30}$$

Having defined $\boldsymbol{\xi} = [\boldsymbol{x}; \boldsymbol{v}]$ and thus $\dot{\boldsymbol{\xi}} = [\dot{\boldsymbol{x}}; \dot{\boldsymbol{v}}]$, Eq. (4.30) reduces to $\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi})$, and therefore can be learned with the methods presented in this section. We verify the performance of both methods in learning a second order motion via a robot task. In this experiment, the iCub performs a loop motion with its right hand, where the motion lies in a vertical plane and thus contains a self intersection point (see Fig. 4.22). Here the task is shown to the robot five times. The motion is learned with seven Gaussian functions with SEDS-Likelihood. The results demonstrate the ability of SEDS to learn second order dynamics.

By extension, since any n-th order autonomous ODE can be transformed into a first-order autonomous ODE, the proposed methods can also be used to learn higher order dynamics; however, at the cost of increasing the dimensionality of

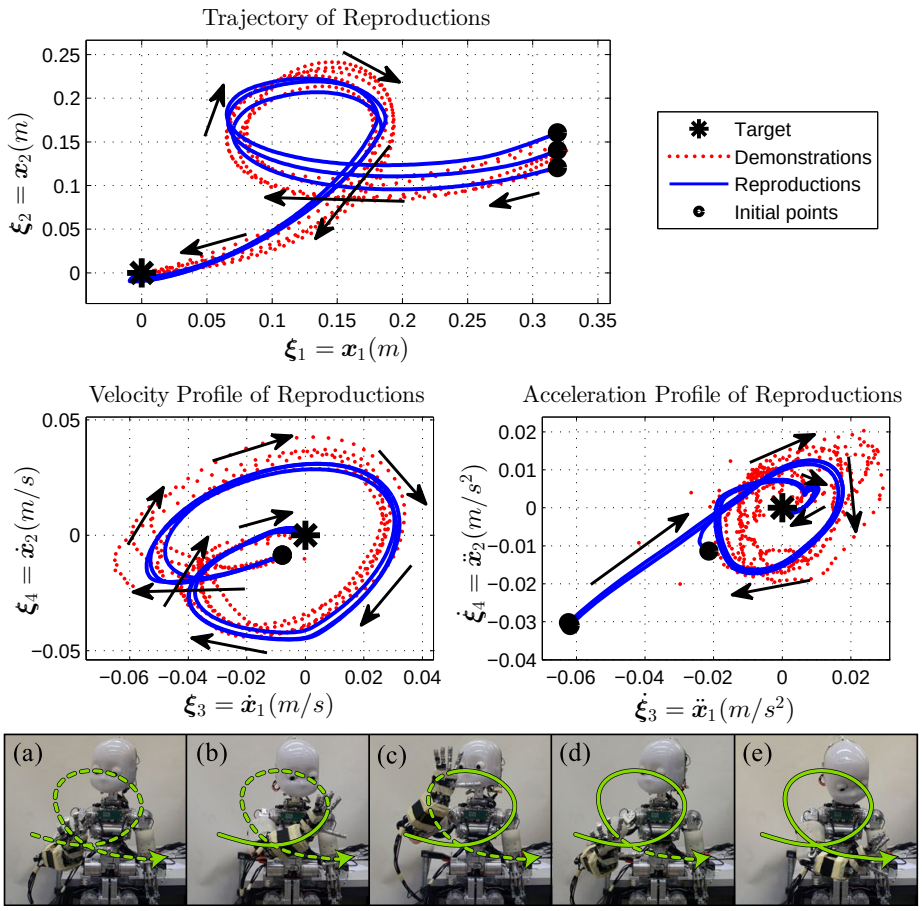


Figure 4.22: Learning a self intersecting motion with a second order dynamics.

the system. If the dimensionality of an n-th order DS is d, the dimensionality of the transformed dynamics into a first order DS is $n \times d$. Hence, increasing the order of the DS is equivalent to increasing the dimensionality of the data. As the dimension increases, the number of optimization parameters also increases. If one optimizes the value of these parameters based on using a quasi-Newton method, the learning problem indeed becomes intractable as the number of dimensions increases. As an alternative solution, one can define the loop motion in terms of both the Cartesian position $\boldsymbol{x}$ and a phase variable. The phase dependent DS has lower dimension (i.e. dimensionality of $d+1$) compared to the second order DS, and is more tractable to learn (we will show an example of such approach in Section 4.6.5). However, as it is already discussed in Section 3.3.2, use of the phase variable makes the system time-dependent. Depending on the application, one may prefer to choose the system of Eq. (4.30) and learn a more complex DS, or to use its phase variable form which is time-dependent, but easier to learn.

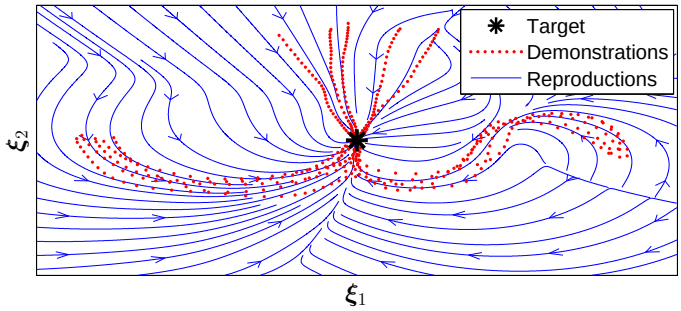


Figure 4.23: Embedding different ways of performing a task in one single model. The robot follow an arc, a sine, or a straight line starting from different points in the workspace. All reproductions were generated in simulation.

4.5.2.4 Encoding Several Motions into one Single Model

We have so far assumed that a single dynamical system drives a motion; however, sometimes it may be necessary to execute a single task in different manners starting from different areas in the space, mainly to avoid joint limits, task constraints, etc. We have shown an example of such an application in an experiment with the Katana-T robot (see Fig. 4.16). Now we show a more complex example and use SEDS to integrate different motions into one single model (see Fig. 4.23). In this experiment, the task is learned using $K = 7$ Gaussian functions, and the 2D demonstrations are collected from pen input using a Tablet-PC. The model is learned using SEDS-Likelihood and is provided with all demonstration data-points at the same time without specifying the dynamics they belong to. Looking at Fig. 4.23 we see that all the three dynamics are learned successfully with a single model and the trajectories are able to approach the target following an arc, a sine function, or a straight line path, starting from the left, right, or top-side of the task space, respectively. While reproductions follow locally the desired motion around each set of demonstrations, they could switch from one motion to another in areas between demonstrations.

4.5.3 Discussion and Conclusion

In this section we presented a method for learning arbitrary discrete motions by modeling them as nonlinear autonomous DS. We proposed a method called *SEDS* to learn the parameters of a GMM by solving an optimization problem under strict stability constraint. We proposed two objective functions *SEDS-MSE* and *SEDS-Likelihood* for this optimization problem. The models result from optimizing both objective functions benefit from the inherent characteristics of autonomous DS, i.e. online adaptation to both temporal and spatial perturbations. However, each objective function has its own advantages and disadvantages. Using log-likelihood is advantageous in that it is slightly more accurate and smoother than MSE. In contrast, the MSE objective function requires fewer parameters than the likelihood one, which may make the algorithm faster in higher dimensions or when a higher number of components is used.

None of the two methods are globally optimal as they deal with a non-convex optimization problem. However, in practice, in the experiments that were reported here, we found that SEDS approximation was quite accurate. The stability conditions at the basis of SEDS are sufficient conditions to ensure global asymptotic stability of nonlinear motions when modeled with a mixture of Gaussian functions. Although our experiments showed that a large library of robot motions can be modeled while satisfying these conditions, these global stability conditions might be too stringent to accurately model some complex motions (we will elaborate more on this limitation of SEDS in Section 4.7).

While in Section 4.5.2.3 we showed how higher order dynamics can be used to model more complicated movements, determining the model order is definitely not a trivial task. It relies on having a good idea of what matters for the task at hand. For instance, higher order derivatives are useful to control for smoothness, jerkiness, energy consumption and hence may be used if the task requires optimizing for such criteria.

Online learning is often crucial to allow the user to refine the model in an interactive manner. At this point in time, the SEDS training algorithm does not allow for online retraining of the model. If one was to add new demonstrations after training the model, one would have to either retrain entirely the model based on the combined set of old and new demonstrations or build a new model from the new demonstrations and merge it with the previous model[11]. For a fixed number of Gaussian functions, the former usually results in having a more accurate model, while the latter is faster to train.

4.6 Stable Estimator of Dynamical Systems-II

In previous sections, we presented two techniques that can be used to ensure local and global stability of autonomous nonlinear DS at the target. These methods use GMR formulation to encode the DS model of the task. However, as outlined in Section 2.2, there exist numerous regression techniques for estimating nonlinear DS, each of which has it own pros and cons. For instance, GPR is a very accurate method but is computationally expensive. GMR is computationally fast and relatively good at extrapolation, but it is not good for online learning. LWPR is a powerful tool for incremental/online learning and computationally fast but it is sensitive to initial parameters and on average requires more parameters than, for example, GMR. When modeling robot motions with DS, it would be advantageous if one could choose the most appropriate regression technique based on the requirements of the task at hand.

[11] Two GMM with K^1 and K^2 number of Gaussian functions can be merged into a single model with $K = K^1 + K^2$ Gaussian functions by concatenating their parameters, i.e.: $\boldsymbol{\theta} = \{\boldsymbol{\theta}^1..\boldsymbol{\theta}^{K^1}..\boldsymbol{\theta}^K\}$, where $\boldsymbol{\theta}^k = \{\pi^k, \boldsymbol{\mu}^k, \boldsymbol{\Sigma}^k\}$. The resulting model is no longer (locally) optimal; however, it could be an accurate estimation of both models, especially when there is no or slight overlapping between the two models.

The two techniques that we have presented so far are constrained by the choice of nonlinear modeling as their stability conditions are expressed in terms of the means and covariances of the mixture model. In this section we extend our previous approach and present a new method, called *SEDS-II*, that can ensure stability of nonlinear DS independently from the choice of the regression model. SEDS-II can also guarantee the stability of a combination of two or more regression techniques, which could allow combining the advantages of different techniques to satisfy requirements of complex tasks. Furthermore, being able to ensure the stability of both autonomous and non-autonomous DS, SEDS-II allows choosing the most appropriate DS formulation for a given task. In addition to the above features, SEDS-II is grounded on less conservative stability conditions compared to SEDS, and thus is able to encode more complex robot motions (we will provide an in-depth comparison between BM, SEDS, and SEDS-II in Section 4.7).

The rest of this section is structured as follows: Section 4.6.1 describes the SEDS-II formulation to generate robot motions. Section 4.6.2 formalizes a constrained optimization problem to estimate a metric of stability. Section 4.6.3 explains an approach to parameterize the metric of stability. Section 4.6.4 provides a formulation to ensure stability of DS-based discrete robot motions. Section 4.6.5 presents experimental results, and Section 4.6.6 concludes the section.

4.6.1 Revisiting DS-Based Formulation

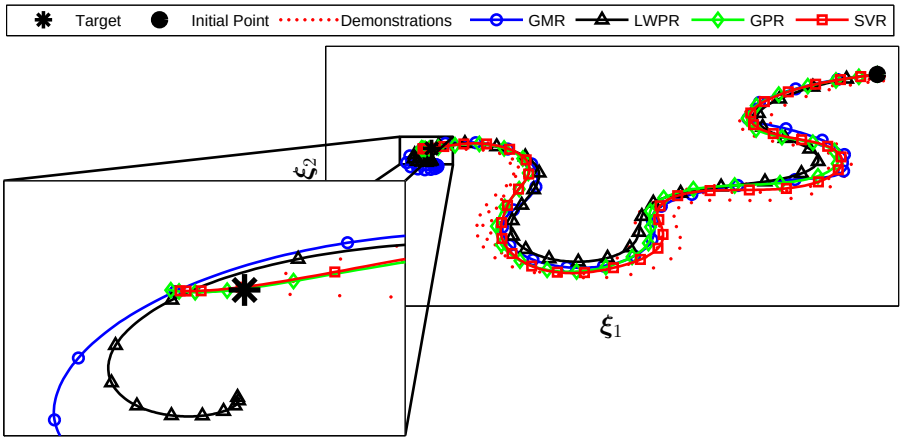
So far, we have only considered a category of DS that is formulated based on GMR according to Eq. (4.8). However, as outlined in Section 2.1, DS-based approaches to generate robot motions can be generally divided into two categories: autonomous and non-autonomous dynamical systems. As we have already observed in previous sections, in autonomous DS the evolution of the state variable $\boldsymbol{\xi}$ only depends on its current value. In contrast in non-autonomous DS, both the current time and the current value of $\boldsymbol{\xi}$ can affect the evolution of the motion, i.e.:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t, \boldsymbol{\xi}), \qquad f : \mathbb{R}^+ \times \mathbb{R}^d \mapsto \mathbb{R}^d \qquad \text{Non-Autonomous DS} \tag{4.31a}$$

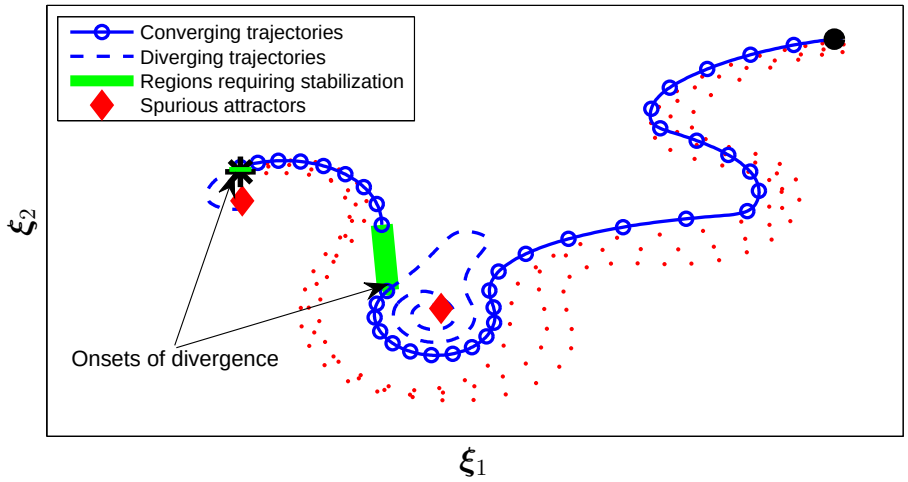
$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}), \qquad f : \mathbb{R}^d \mapsto \mathbb{R}^d \qquad \text{Autonomous DS} \tag{4.31b}$$

where $\boldsymbol{f}(.)$ is a continuous function. We will henceforth use the notation $\boldsymbol{f}(.)$ to refer to both autonomous and non-autonomous DS.

The method that we present in this section is inspired from the following observation: "In a region close to demonstrations, the estimates from $\boldsymbol{f}(.)$ mostly exhibit a converging behavior to the target." For instance, Fig. 4.24a shows an example of unstable estimates of a motion that are learned from four of the best to date regression techniques. One can clearly see that the generated trajectories closely follow the demonstrations, and then suddenly diverge from the target.



(a) As can be seen in this example, generated trajectories closely follow the demonstrations in the largest part of the motion, and diverge from the target only in some areas.



(b) Illustration of divergence regions when using GMR to generate a trajectory from the depicted initial point. The motion only requires stabilization in a small region that is highlighted in green.

Figure 4.24: Illustration of convergence and divergence behaviors of a two-dimensional dynamics estimated on the basis of three training examples using GMR, LWPR, GPR, and SVR. Please refer to Section 4.6.1 for further information.

In contrast to our previous approaches that explicitly put constraints on $\boldsymbol{f}(\boldsymbol{\xi})$ to make it asymptotically stable at the target, here we seek an alternative approach: We use the estimated DS from a regression technique to derive the motion as long as it shows converging behavior. Only in the case of divergence, the estimations from the regression are corrected so that the convergence behavior is retained throughout the state space. In other words, we are interested in determining a stabilizing command $\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.)) \in \mathbb{R}^d$ such that the resulting DS:

$$\dot{\boldsymbol{\xi}} = \tilde{\boldsymbol{f}}(.) = \boldsymbol{f}(.) + \boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.)) \tag{4.32}$$

is 1) *globally asymptotically stable* at the target $\boldsymbol{\xi}^*$ to ensure the convergence of all trajectories to the target, and 2) an *accurate estimation of the user demonstrations* to satisfy the task requirements. Both of the above requirements are essential for $\tilde{\boldsymbol{f}}(.)$ to provide a useful control policy. Note that in the above formulation, $\boldsymbol{u}$ is zero when $\boldsymbol{f}(.)$ is converging, and it only becomes nonzero once $\boldsymbol{f}(.)$ diverges. One should be careful when generating $\boldsymbol{u}$ to avoid discontinuities in the output of the system when $\boldsymbol{f}(.)$ switches from converging to diverging behavior and vice versa.

Fig. 4.24b illustrate through an example our desired control policy that is just described above. In this example, the DS $\boldsymbol{f}(.)$ is estimated with GMR. Starting from the depicted initial point, the DS initially exhibits converging behavior. However, after some iterations it diverges from the target (i.e. exhibits divergence behavior) and approaches a spurious attractor. By identifying the regions where $\boldsymbol{f}(.)$ shows divergence behaviors, we could correct the motion by applying the stabilizing command. After a few iterations, the divergence behavior vanishes, and the motion can solely be derived from $\boldsymbol{f}(.)$ without any need to apply the stabilizing command until it reaches a region close to the target, where it diverges for the second time. Using the stabilizing command, the motion can successfully reach the target.

The described strategy requires a so-called metric of stability to detect when $\boldsymbol{f}(.)$ is exhibiting diverging behaviors from the target. Finding this metric is non-trivial as it could differ from one example to another. However, we could exploit the demonstrations of the task to build an estimate of the metric of stability suited for the task at hand. We then exploit this metric to identify the unstable regions, and to devise a control strategy to generate the stabilizing command $\boldsymbol{u}$ so as to ensure global asymptotic stability of $\tilde{\boldsymbol{f}}(.)$ at the target.

Fig. 4.25 shows the schematic of the control flow for the new system. First, note the difference between the stabilizing command $\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.))$ and the robot control command $\boldsymbol{\tau}$. The former is a virtual signal that is generated at the kinematic level to make the DS planer stable, while the latter is the actual torque or force that is applied to the robot to follow the desired motion. In this control architecture, the stability of the system is ensured while executing the task, as opposed to BM and SEDS that ensure stability during training.

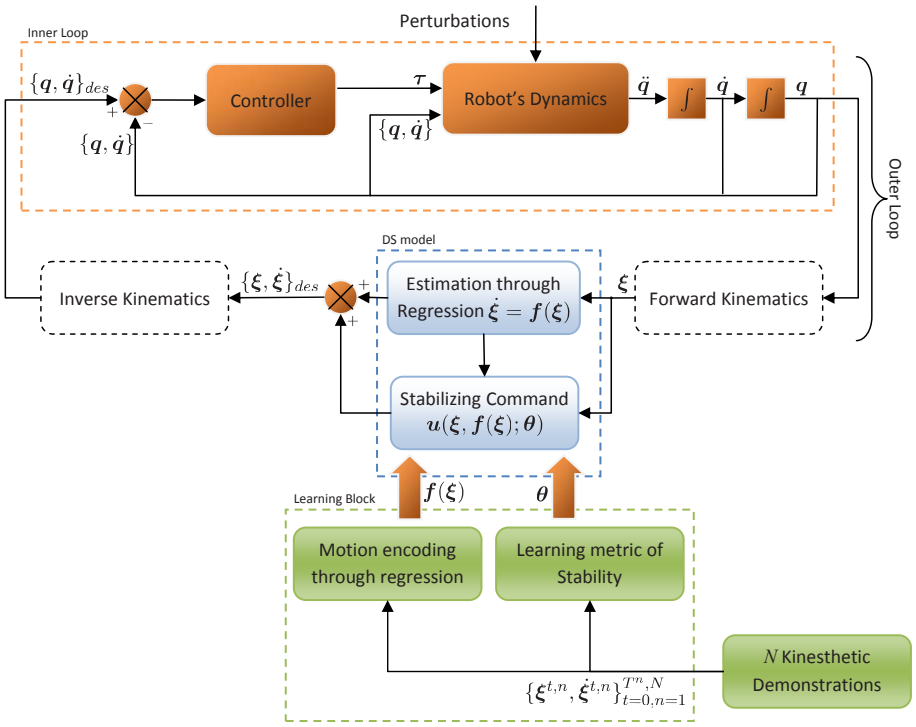


Figure 4.25: The system's architecture illustrating the control flow when using SEDS-II formulation to derive the stable control policy. In this graph, $\boldsymbol{q}$, $\boldsymbol{\tau}$, and $\boldsymbol{\xi}$ correspond to the robot's joint angles, joint torques, and the state variables describing the robot motion, respectively. Note that here we assume $\boldsymbol{f}$ is an autonomous system. For non-autonomous systems, one should also add the time variable as an input to the DS block.

In the motion encoding block, an estimate of DS model is constructed based on demonstrations using any smooth regression model such as GMR, GPR, LWPR, etc. Depending on the regression method, the motion encoding step may be done offline or online. In the present approach, estimation of the metric of stability can solely be done offline.

The presented approach is in spirit identical to the Control Lyapunov Function (CLF)-based control scheme in control engineering (Artstein, 1983; Primbs et al., 1999; Jiang et al., 2009; Sontag, 1998). The construction of a CLF is nontrivial, and has only been solved for special classes of DS (Kokotovic & Arcak, 2001). In contrast, in our approach we build an estimate of the metric of stability (the equivalence of CLF) from demonstrations. Furthermore, as our approach is solely applied at the kinematic level, it is tailored to generate stabilizing commands that modifies the unstable DS given by $\boldsymbol{f}(.)$ as least as possible at each iteration. Hence, it tries to maximize the similarity between the stabilized and the unstable DS which is crucial for our application.

Apart from the CLF approach, there are also a number of other nonlinear control techniques that deal with the problem of generating the control command $\boldsymbol{u}$ so as to stabilize the nonlinear DS that is given by Eq. (4.32) at the target. However, most of these approaches do not consider the requirement of

"stabilizing $\boldsymbol{f}$ at the target, while generating motions that resemble the user demonstrations" which is essential in our implementation. Nevertheless, the Optimal Control technique (Bryson & Ho, 1975) and its sub-branches such as the Model Predictive Control (Kulchenko & Todorov, 2011) can be formulated so as to fulfill the above requirement. Despite the successful realtime implementation of these approaches for linear DS (Boyd & Vandenberghe, 2004), their implementation for nonlinear DS is still an open question and realtime solutions only exist for particular cases.

4.6.2 Learning Metric of Stability

As outlined in the previous section, in order to ensure global stability of $\tilde{\boldsymbol{f}}(.)$ at the target $\boldsymbol{\xi}^*$, we need to determine a metric of stability that can be used as a gauge to detect divergence behaviors. The energy of the system can be considered as a good candidate for this purpose. Let us retake the example presented in Fig. 4.24b, and assume the energy of the system is defined by a function $V(\boldsymbol{\xi})$. Figure 4.26 illustrates the energy levels of $V(\boldsymbol{\xi})$ (note that the energy function is positive everywhere except at the target where it vanishes).

Following our example, at the initial point the system has the energy $V(\boldsymbol{\xi}^0)$. As the trajectory evolves, its energy continuously dissipates until it reaches the first divergence point (indicated with a red triangle). At this point, the energy of the system starts increasing which eventually yields convergence to a spurious attractor. In order to avoid this behavior, the estimate from the GMR should be corrected so that the energy of the motion keeps decreasing. The regions that require such corrections are indicated by a green line in Fig. 4.26.

Two observations follow from the above example: 1) Divergence behavior can be observed in regions $\mathbb{R}^d \setminus \boldsymbol{\xi}^*$ where the energy of the system no longer decreases, 2) The correct determination of divergence points strongly depends on the way the metric of stability (i.e. the energy function) is defined. The latter is a crucial factor as an improper metric of stability could yield errors in determining the divergence points, which subsequently leads to unnecessary estimation errors in following the demonstrations[12].

To avoid this issue, we take an imitation learning perspective and suggest a procedure to build a valid estimate of the energy function from the demonstrations. We call the function $V(\boldsymbol{\xi}) > 0$ a valid energy function based on the user demonstrations if as one moves along any of the demonstration trajectories its energy decreases and finally extinguishes at $\boldsymbol{\xi}^*$:

$$V(\boldsymbol{\xi}^{t,n}) > V(\boldsymbol{\xi}^{t+1,n}) \qquad \forall t \in 0..T^n - 1,\ n \in 1..N \tag{4.33}$$

[12] As we show later on in Section 4.6.6, an improper metric of stability does not eliminate the global asymptotic stability of $\tilde{\boldsymbol{f}}(.)$. However, it can reduce the accuracy in estimation of the user demonstrations which is a crucial factor in our implementation.

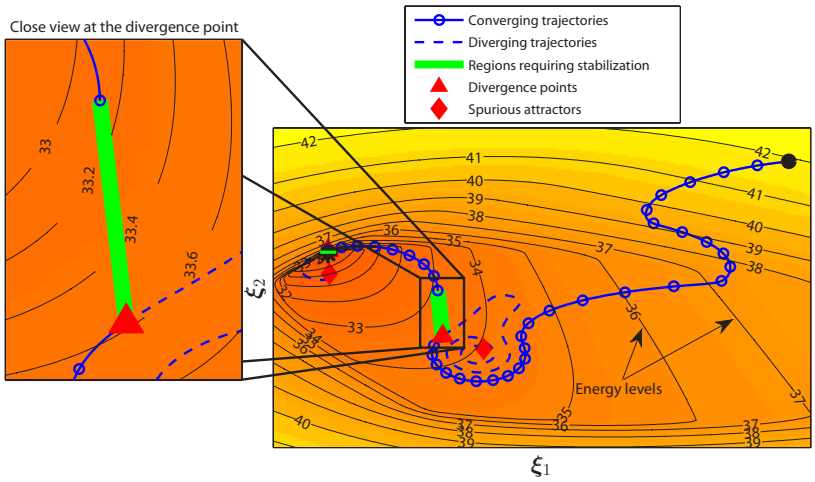


Figure 4.26: The system's architecture illustrating the control flow when using SEDS-II formulation to derive the stable control policy.

The above criterion is equivalent to request that the energy decreases over time. This can be verified by computing the scalar product between the energy gradient and the velocity at each training data point except at the target:

$$\begin{aligned}\dot{V}(\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}) &= \frac{dV(\boldsymbol{\xi}^{t,n})}{dt} = \Big(\frac{dV(\boldsymbol{\xi}^{t,n})}{d\boldsymbol{\xi}}\Big)^T \frac{d\boldsymbol{\xi}^{t,n}}{dt} \\ &= \big(\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}^{t,n})\big)^T \dot{\boldsymbol{\xi}}^{t,n} < 0 \end{aligned} \tag{4.34}$$

where $(.)^T$ denotes the transpose.

In this section, we propose a constrained optimization problem that can be used to obtain an estimate of the energy function from the user demonstrations. To reach this goal, we first proceed by parameterizing the energy function with a vector of parameters $\boldsymbol{\theta}$ and denote it with $V(\boldsymbol{\xi}; \boldsymbol{\theta})$. More information on parameterization of V will be discussed later on in Section 4.6.3. Next, in order to ensure that the energy function has a single global minimum, we request that: 1) V is positive $\forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^*$, 2) It is zero at the target $\boldsymbol{\xi}^*$, and 3) Its Hessian $\nabla_{\boldsymbol{\xi}\boldsymbol{\xi}} V(\boldsymbol{\xi}; \boldsymbol{\theta})$ is a positive definite matrix $\forall \boldsymbol{\xi} \in \mathbb{R}^d$.[13] The above three requirements naturally yield the vector of partial derivatives vanishes at the target, i.e. $\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}^*; \boldsymbol{\theta}) = 0$. We formulate these strict conditions as the constraints of the optimization problem. Finally, we define the optimization objective function so that it maximizes the number of training data points that satisfy Eq. (4.34). The more the training data points that satisfy Eq. (4.34), the better the estimated energy function represents the demonstrations.

It should be noted that the conditions given by Eq. (4.34) cannot be considered in the optimization constraints as it might not be possible to ensure them

[13] Note that the requirement on the positive definiteness of $\nabla_{\boldsymbol{\xi}\boldsymbol{\xi}} V(\boldsymbol{\xi}; \boldsymbol{\theta})$ is a conservative condition. However, as we will show later in Section 4.6.3, using this condition is advantageous in that it can be easily satisfied via choosing a proper energy parameterization, yielding to a significantly faster training algorithm.

for all the training data points. This is partially due to the fact that the demonstrations are noisy observations of instances of the DS, and partially due to the way in which the energy function is parameterized. The former is inevitable as some of the datapoints may even contradict each other by having, for example, different velocity profiles at the same point. The latter directly controls the amount of nonlinearity that can be captured with V, and thus its effect can be reduced or eliminated by choosing a more complex parameterization for the energy function[14]. We will elaborate more about the parameterization of V in Section 4.6.3.

Putting together all the points described above, a locally optimal solution to $\boldsymbol{\theta}$ and thus by construction the energy function can be found by solving the following constrained optimization problem:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{(1+\bar{w})\text{sign}(\psi^{t,n}) + (1-\bar{w})}{2} (\psi^{t,n})^2 \tag{4.35}$$

subject to

$$V(\boldsymbol{\xi}; \boldsymbol{\theta}) > 0 \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^* \tag{4.36a}$$

$$V(\boldsymbol{\xi}^*; \boldsymbol{\theta}) = 0 \tag{4.36b}$$

$$\nabla_{\boldsymbol{\xi}\boldsymbol{\xi}} V(\boldsymbol{\xi}; \boldsymbol{\theta}) \succ 0 \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \tag{4.36c}$$

where $\succ$ denotes the positive definiteness of a matrix, $\bar{w}$ is a small positive scalar (i.e. $\bar{w} > 0$ and $\bar{w} \ll 1$), and and $\psi^{t,n}$ is:

$$\psi^{t,n} = \psi(\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}; \boldsymbol{\theta}) = \frac{(\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}^{t,n}))^T \dot{\boldsymbol{\xi}}^{t,n}}{\|\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}^{t,n}; \boldsymbol{\theta})\| \|\dot{\boldsymbol{\xi}}^{t,n}\|} \tag{4.37}$$

Throughout this section we use the interior-point algorithm to solve this optimization problem (see Section 2.3). Note that by defining the objective function as described above, the optimization problem favors lowering the number of data points for which Eq. (4.34) does not hold, and as a second priority, it tries to align the gradient of energy with the negative direction of movement. By tuning the value of $\bar{w}$, one can control the priority portion of these two objectives. Furthermore, the normalization by the norm of the gradient and velocity vectors in Eq. (4.37) is essential to give an equal importance to all training data points during the optimization.

It now remains to find a proper parameterization of the energy function. Note that, as we will show later on, the global asymptotic stability of $\tilde{\boldsymbol{f}}(.)$ can be ensured irrespective of the number of data points for which Eq. (4.34) does not hold. However, the lower the number of data points that violate Eq. (4.34), the less error in determining divergence points of $\boldsymbol{f}(.)$.

[14] As for analogy, consider the example of fitting a polynomial on a set of datapoints that are sampled from a nonlinear function. In this example, the choice of the order of polynomials directly affect the accuracy with which the underlying function is estimated.

4.6.3 Energy Function Parameterizations

One of the main constraints when estimating an energy function is the requirement of nonnegativity throughout the state space (see Eqs. (4.36a) and (4.36c)). This verification is practically non-trivial (if not impossible) and requires a significant amount of computational power. To illustrate this issue, let us consider that the working space of a robot is meshed uniformly with m grids in each dimension which results in having m^d grids and by extension constraints to verify Eq. (4.36a). The condition given by Eq. (4.36c) also yields $d \times m^d$ constraints[15]. Thus in the optimization problem defined above, the number of constraints that should be satisfied are $(d+1)m^d + 1$. Needless to say, even with a dense mesh, it is still not possible to guarantee that the constraints on the energy function are satisfied throughout the state space.

However, the problem of verification can be bypassed by considering some special forms for the energy function. In this section we propose a new parameterization, called Weighted Sum of Asymmetric Quadratic Functions (WSAQF), that could naturally bypass the verification problem. Note that the energy function parameterization is not limited to the above method, and one can opt for other alternatives depending on the task at hand[16].

As it appears from its name, the WSAQF parametrization models the energy function as a weighted sum of asymmetric quadratic functions:

$$V(\boldsymbol{\xi};\boldsymbol{\theta}) = (\boldsymbol{\xi}-\boldsymbol{\xi}^*)^T \boldsymbol{P}^0 (\boldsymbol{\xi}-\boldsymbol{\xi}^*) + \sum_{\ell=1}^{\mathscr{L}} \beta^\ell(\boldsymbol{\xi};\boldsymbol{\theta}) \Big((\boldsymbol{\xi}-\boldsymbol{\xi}^*)^T \boldsymbol{P}^\ell (\boldsymbol{\xi}-\boldsymbol{\mu}^\ell-\boldsymbol{\xi}^*) \Big)^2 \quad (4.38)$$

where $\mathscr{L}$ is the number of asymmetric quadratic functions defined by the user, $\boldsymbol{\mu}^\ell \in \mathbb{R}^d$ are vectors influencing the asymmetric shape of the energy function, $\boldsymbol{P}^\ell \in \mathbb{R}^{d\times d}$ are positive definite matrices, and the coefficients $\beta^\ell(\boldsymbol{\xi};\boldsymbol{\theta})$ are:

$$\beta^\ell(\boldsymbol{\xi};\boldsymbol{\theta}) = \begin{cases} 1 & \forall \boldsymbol{\xi} : (\boldsymbol{\xi}-\boldsymbol{\xi}^*)^T \boldsymbol{P}^\ell (\boldsymbol{\xi}-\boldsymbol{\mu}^\ell-\boldsymbol{\xi}^*) \geq 0 \\ 0 & \forall \boldsymbol{\xi} : (\boldsymbol{\xi}-\boldsymbol{\xi}^*)^T \boldsymbol{P}^\ell (\boldsymbol{\xi}-\boldsymbol{\mu}^\ell-\boldsymbol{\xi}^*) < 0 \end{cases} \qquad \forall \ell \in 1..\mathscr{L} \quad (4.39)$$

The learning parameters of WSAQF are the components of the matrices $\boldsymbol{P}^\ell$ and vectors $\boldsymbol{\mu}^\ell$, i.e. $\boldsymbol{\theta} = \{\boldsymbol{P}^0, ..., \boldsymbol{P}^{\mathscr{L}}, \boldsymbol{\mu}^1, ..., \boldsymbol{\mu}^{\mathscr{L}}\}$. In this formulation, the optimization constraints given by Eq. (4.36) are satisfied automatically by only requiring $\boldsymbol{P}^\ell \succ 0$, $\forall \ell = 0..\mathscr{L}$.[17]

[15] The constraint on the positive definiteness of $\nabla_{\boldsymbol{\xi}\boldsymbol{\xi}} V(\boldsymbol{\xi};\boldsymbol{\theta})$ needs to be transformed into requiring its d eigenvalues to be positive. Thus, at each mesh point, d constraints should be checked yielding $d \times m^d$ constraints

[16] For example, one can use the Sum of Squares (SOS) approach (Parrilo, 2000) to build an estimate of $V(\boldsymbol{\xi};\boldsymbol{\theta})$. However, we do not consider this approach here as it is mainly devised for the cases for which the DS $\boldsymbol{f}(.)$ is expressed as a polynomial so as to avoid numerical verification of the optimization's constraints over a mesh.

[17] This can be verified as follows:

1. By substituting $\boldsymbol{\xi} = \boldsymbol{\xi}^*$ into Eq. (4.38), we obtain $V(\boldsymbol{\xi}^*;\boldsymbol{\theta}) = 0$.
2. The positive definiteness of $\boldsymbol{P}^0$ implies that the first term in Eq. (4.38) is positive $\forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^*$. Furthermore, the second term in Eq. (4.38) is always nonnegative. Hence we have $V(\boldsymbol{\xi};\boldsymbol{\theta}) > 0$, $\forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^*$.
3. The positive definiteness of all the matrices $\boldsymbol{P}^\ell$ yields a positive definite Hessian matrix $\nabla_{\boldsymbol{\xi}\boldsymbol{\xi}} V(\boldsymbol{\xi};\boldsymbol{\theta}) \succ 0$, $\forall \boldsymbol{\xi} \in \mathbb{R}^d$.

Thus, the optimization problem given by (4.35) reduces to:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{(1+\bar{w})\text{sign}(\psi^{t,n}) + (1-\bar{w})}{2} (\psi^{t,n})^2 \tag{4.40}$$

subject to

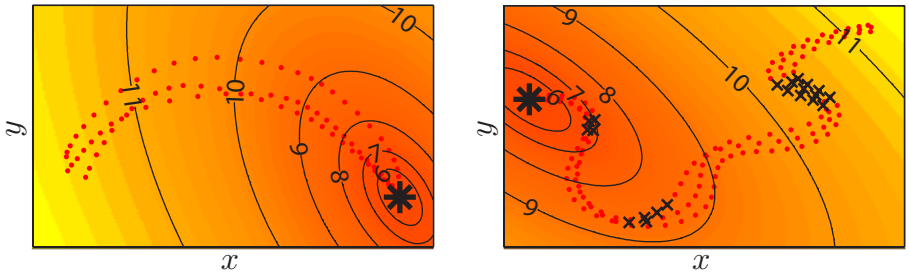
$$\boldsymbol{P}^{\ell} \succ 0 \qquad \forall \ell = 0..\mathscr{L} \tag{4.41}$$

The use of WSAQF parameterization significantly reduces the workload of the optimization. In contrast to the original problem, verifying only the positive definiteness of P^{ℓ} in WSAQF parameterization requires $(\mathscr{L}+1)d$ constraints.

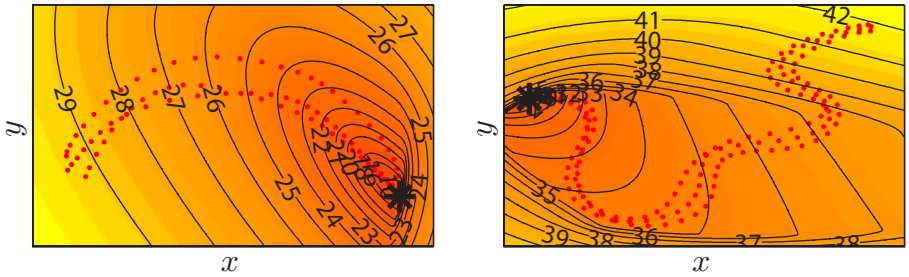
One concern that is usually associated with solving the above optimization problem is the ambiguity in defining $V(\boldsymbol{\xi})$: "if $V(\boldsymbol{\xi})$ is a Lyapunov function, its arbitrary scalar multiplication $cV(\boldsymbol{\xi})$, $c > 0$, is also a Lyapunov function". However, in our implementation, this is not problematic as we solely work with the normalized value of the gradient of the energy function both when estimating the energy function (see Eq. (4.37)) and when generating the stabilizing command (see Section 4.6.4). Thus, the solution of the above optimization problem, and in general the result from SEDS-II method is invariant to the value of c.

An especial case of the WSAQF parameterization happens when $\mathscr{L} = 0$. In this case, V corresponds to the well known Quadratic Energy Function (QEF). Compared to the general case, QEF assumes the energy function is elliptic around the origin. Figure 4.27 shows examples of energy functions that are learned from a set of user demonstrations using the WSAQF and its especial form the QEF parameterizations. Here we report on two 2D motions: The first example is a single-curve 2D motion, and the second one is a multi-curve motion. Figure 4.27a shows the obtained energy function for the special case of the QEF parameterization. Figure 4.27b illustrates the results that are estimated using the general WSAQF parameterization with $\mathscr{L} = 1$ and $\mathscr{L} = 3$ for the single-curve and multi-curve motions, respectively. In this figure, the training data points that do not satisfy Eq. (4.34) are marked with a cross. As can be seen, all training data points of the single-curve motion satisfy Eq. (4.34) in both approaches. For the multi-curve motion, there are some points that do not satisfy this equation when using the QEF parameterization. However, the general form of WSAQF is able to model more complex energy function so that all the points satisfy Eq. (4.34).

The proper choice of the number of asymmetric functions is important. The higher the $\mathscr{L}$, the more complex energy function that can be modeled; however, at the cost of having more optimization parameters. Thus, the complexity of optimization increases by increasing $\mathscr{L}$. Furthermore by using a large number of asymmetric functions, one may overfit the nonlinearities due to the noise in the demonstrations. Thus, there is a compromise inherent in setting the value of $\mathscr{L}$. In the experiments we present here, we proceed as follows: we start with $\mathscr{L} = 0$, i.e. the QEF parametrization, and incrementally increase $\mathscr{L}$ until it no



(a) The QEF parameterization



(b) The WSAQF parameterization with $\mathscr{L} = 1$ and $\mathscr{L} = 3$ for the left and right motions, respectively.

Figure 4.27: Two examples of estimating the energy function from a set of user demonstrations. The contour curves of the energy functions are shown with black solid lines. For clarity of the graph, we show the contours of $\log(V)$. The background color also illustrates the energy level. The lighter the color, the higher the energy. The black star indicates the target point, training data points are shown with red dots, and those data points that do not satisfy Eq. (4.34) are marked with a cross. Please refer to Section 4.6.3 for more details.

longer affects or marginally improves the accuracy in estimation of the training data points.

Figure 4.28 shows such evaluation for the two motions presented in Fig. 4.27. As we can see, for the single-curve motion, the increase in $\mathscr{L}$ does not affect the estimation accuracy. Thus the QEF parameterization, i.e. $\mathscr{L} = 0$ is the proper energy model for this motion. For the multi-curve motions, the accuracy in estimation improves by increasing $\mathscr{L}$ from 0 to 2. For $\mathscr{L} = 3$, we could still observe a slight improvement in the accuracy. However, as this improvement is very negligible, one may prefer to opt for the energy model with $\mathscr{L} = 2$.

4.6.4 Computation of Stabilizing Command

In Section 4.6.2 we proposed an optimization problem to compute a valid energy function based on the user demonstrations. Aside from using the energy function to detect diverging behaviors, we could exploit it to derive the stabilizing command $\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.))$ so as to ensure global asymptotic stability of $\tilde{\boldsymbol{f}}(.)$ at the target $\boldsymbol{\xi}^*$. To simplify the notation, we define:

$$\bar{\boldsymbol{v}}(\boldsymbol{\xi};\boldsymbol{\theta}) = \frac{\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi};\boldsymbol{\theta})}{\|\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi};\boldsymbol{\theta})\|} \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \backslash \boldsymbol{\xi}^* \tag{4.42}$$

$$\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.);\boldsymbol{\theta}) = \bar{\boldsymbol{v}}(\boldsymbol{\xi};\boldsymbol{\theta})^T \boldsymbol{f}(.) \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \backslash \boldsymbol{\xi}^* \tag{4.43}$$

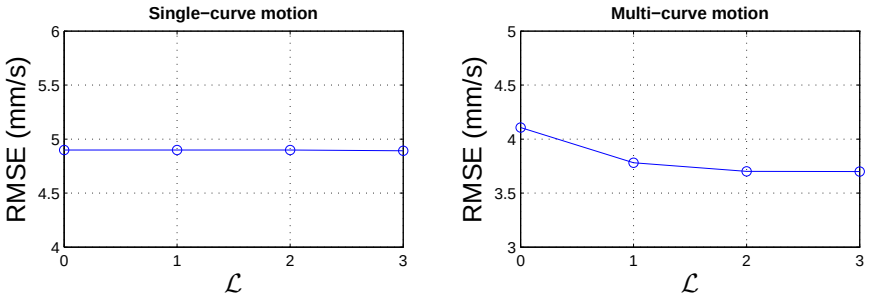


Figure 4.28: The number of asymmetric quadratic functions can be selected as follows: Start with $\mathscr{L} = 0$, and incrementally increase $\mathscr{L}$ until it no longer affects or marginally improves the accuracy in estimation of the training data points. Following this procedure, the proper value of $\mathcal{L}$ for the single and multi-curve motions are 0 and 2.

where $\bar{\boldsymbol{v}}(\boldsymbol{\xi})$ is a unitary vector that is aligned with the gradient of the energy function, and α is a measure of the misalignment between $\bar{\boldsymbol{v}}(\boldsymbol{\xi})$ and the estimated function $\boldsymbol{f}(.)$. The DS $\boldsymbol{f}(.)$ is converging when the estimated flow of motion from $\boldsymbol{f}(.)$ is aligned with the direction that decreases the energy potential. Let us also define:

$$\rho(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta}) = \kappa_{\boldsymbol{\xi}} e^{-\sigma_{\boldsymbol{\xi}} \|\boldsymbol{\xi}\|} + \kappa_{\dot{\boldsymbol{\xi}}} (1 - e^{-\sigma_{\boldsymbol{\xi}} \|\boldsymbol{\xi}\|}) e^{-\sigma_{\dot{\boldsymbol{\xi}}} \|\boldsymbol{f}(.)\|} \tag{4.44}$$

where the parameters $\kappa_{\boldsymbol{\xi}}$, $\kappa_{\dot{\boldsymbol{\xi}}}$, $\sigma_{\boldsymbol{\xi}}$, $\sigma_{\dot{\boldsymbol{\xi}}}$ are positive scalars.

□

Theorem 4.3 *Consider a smooth activation function $\phi(\alpha) \in \mathbb{R}$:*

$$\phi(\alpha) = \begin{cases} 1 & 0 < \alpha \\ 0.5 \cos(\tau \alpha) + 0.5 & -\frac{\pi}{\tau} \leq \alpha \leq 0 \\ 0 & \alpha < -\frac{\pi}{\tau} \end{cases} \tag{4.45}$$

where $\tau > 0$ is a scalar to tune the slope of the activation function (see Fig. 4.29). The DS defined by Eq. (4.32) is globally asymptotically stable at the target $\boldsymbol{\xi}^*$ in $\mathbb{R}^d$ if the stabilizing command $\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta})$ is determined according to:

$$\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.)) = \begin{cases} -\phi\big(\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.))\big)\Big(\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.)) + \rho(\boldsymbol{\xi}, \boldsymbol{f}(.))\Big)\bar{\boldsymbol{v}}(\boldsymbol{\xi}) & \forall \boldsymbol{\xi} \in \mathbb{R}^d \backslash \boldsymbol{\xi}^* \\ -\boldsymbol{f}(.) \qquad \textit{if } \boldsymbol{\xi} = \boldsymbol{\xi}^* \end{cases} \tag{4.46}$$

Proof: See Appendix A.3.

■

Note that for the clarity of Eqs. (4.45) and (4.46), we have denoted $\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta})$ with α, and also dropped the dependence on the parameters $\boldsymbol{\theta}$.

The activation function given by Eq. (4.45) is used to trigger the stabilizing command when the system is at the edge of divergence. Recall that for $\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta}) > 0$, the DS given by $\boldsymbol{f}(.)$ exhibits diverging behaviors (it increases

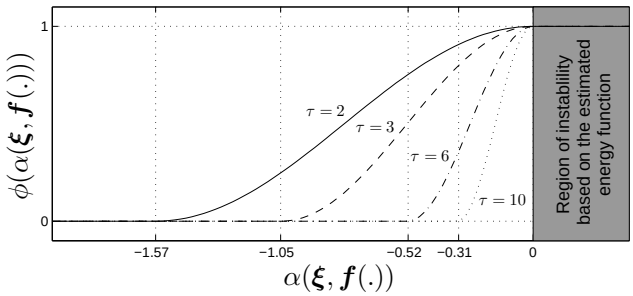


Figure 4.29: Tuning the width of the stability margin with the parameter τ. The lower the τ, the larger the width of the stability margin.

the energy of the system). In these situations $\phi(\alpha) = 1$, and the stabilizing command can be used to ensure the stability of the DS. For $-\pi/\tau \leq \alpha(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta}) \leq 0$, the activation function smoothly rises from 0 to 1. In this region, although $\boldsymbol{f}(.)$ is converging, it is still slightly modified with the stabilizing command to ensure the continuity of $\tilde{\boldsymbol{f}}(.)$. For $\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta}) < \pi/\tau$, the system has a safe stability margin[18] (according to the user preference), hence no stabilizing command is generated, i.e. $\boldsymbol{u}(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta}) = 0$.

Fig. 4.29 illustrates the effect of τ on the activation function. By decreasing τ, the width of the activation region increases. There is a compromise inherent in setting the value of τ. By lowering τ, we could avoid a sudden change in the direction of motion. On the other hand, by setting a high τ, we could keep the larger part of the stable region intact.

Fig. 4.30 illustrates through an example the effect of the first two terms in Eq. (4.46), and the tuning parameters $\kappa_{\boldsymbol{\xi}}$ and $\kappa_{\dot{\boldsymbol{\xi}}}$ on the final stable DS $\tilde{\boldsymbol{f}}(.)$. In this example, the original DS $\boldsymbol{f}(.)$ is locally stable in a unit circle around the origin (see Fig. 4.30a). By only applying the first term in Eq. (4.46), the diverging behavior outside the unit circle transforms into a neutral behavior (see Fig. 4.30b). The converging behavior can be obtained by applying the second term in Eq. (4.46), and subsequently the DS becomes globally asymptotically stable at the target. The tuning parameters $\kappa_{\boldsymbol{\xi}}$ and $\kappa_{\dot{\boldsymbol{\xi}}}$ can be used to set the rate of convergence in unstable regions (see Fig. 4.30c and Fig. 4.30d).

Two observations follow from Fig. 4.30: 1) As we expect, the original DS is modified when it shows unstable behavior according to the estimated energy function (i.e. the metric of stability), and 2) The parts of velocity in the original DS that are not the source of instability are preserved after stabilization. This includes the velocity components that are orthogonal to $\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}; \boldsymbol{\theta})$. Note that in this example we have manually defined the energy function for illustrative purpose. As outlined before, in our experiments the energy function is estimated from a set of training data points.

[18] The stability margin is defined by $SM = -\alpha(\boldsymbol{\xi}, \boldsymbol{f}(.); \boldsymbol{\theta})$ and provides an indication about how far the system is from becoming unstable. When $SM = 0$, the system is at the edge of instability, and $SM < 0$ corresponds to an unstable system.

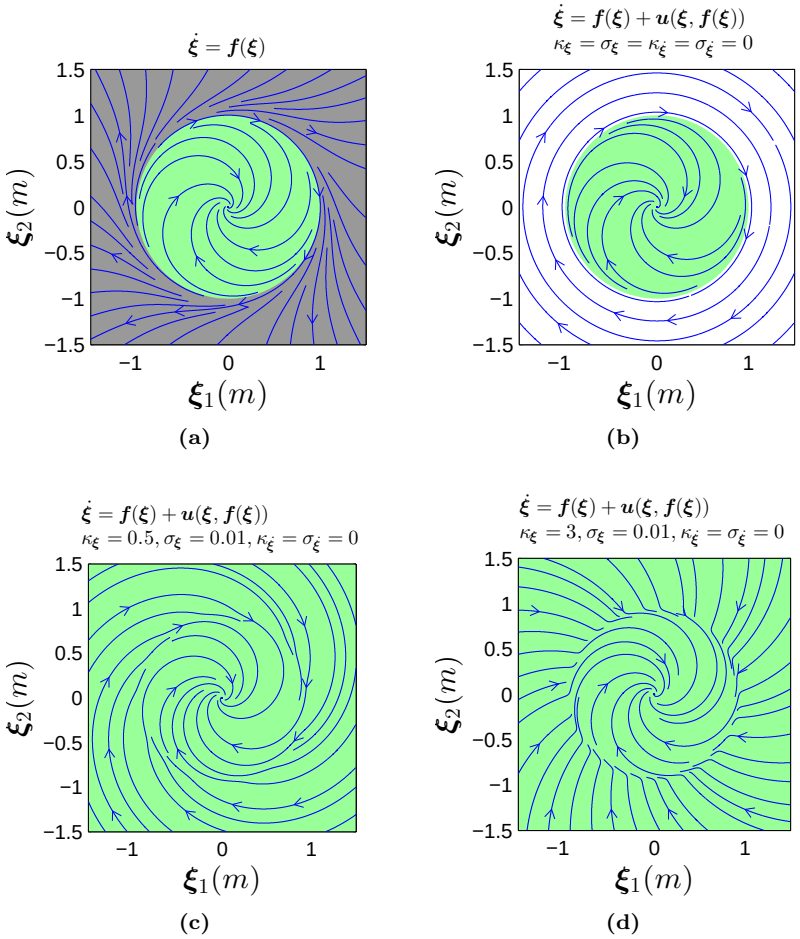


Figure 4.30: An example illustrating the effect of the stabilizing command on a locally stable DS. **(a):** The original DS is defined by $\dot{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_2 + \boldsymbol{\xi}_1(\boldsymbol{\xi}_1^2 + \boldsymbol{\xi}_2^2 - 1)$ and $\dot{\boldsymbol{\xi}}_2 = \boldsymbol{\xi}_1 + \boldsymbol{\xi}_2(\boldsymbol{\xi}_1^2 + \boldsymbol{\xi}_2^2 - 1)$. The green and grey regions represents the stable and unstable regions. In this example, we set $\tau = 20$ and define $V(\boldsymbol{\xi}) = \boldsymbol{\xi}^T\boldsymbol{\xi}$. **(b):** By applying the first term in Eq. (4.46), the diverging behavior is replaced by a neutral behavior (the white region). **(c)-(d):** By using both terms in Eq. (4.46), the neutral behavior changes to converging behavior, and the DS becomes globally asymptotically stable at the target. By increasing $\kappa_{\boldsymbol{\xi}}$, the rate of convergence in the previously unstable region increases.

4.6.5 Experiments

We evaluate the performance of the proposed approach in two ways: 1) On two complex planar motions that are inferred from human demonstrations, and that are described by autonomous and non-autonomous DS. With this experiment, we illustrate one of the main properties of the proposed method that it can be used to stabilize unstable DS that are modeled with different regression techniques and with different types of DS, and 2) in a robot experiment performed on the 7-DoF Barrett WAM arm. In this experiment, we demonstrate that our approach allows combining two different regression techniques in order to benefit from the advantages of both.

The demonstrations of the two planar motions are collected at 50Hz from pen input using a Tablet-PC. The first motion is complex in that it includes two motions that overlap on the $x-y$ plane (see Fig. 4.31a). The second planar motion is composed of trajectories that begin by distancing themselves from the target point, and then approach it (see Fig. 4.31c). For each of these motions, we first start by building an estimate of the energy function from the demonstrations. For the first and the second motions, we model the energy function with the QEF and WSAQF parameterization with $\mathscr{L}=3$, respectively.

In order to successfully learn the first motion and to avoid ambiguity in the overlapping region, we model it as a non-autonomous DS, i.e. $\dot{\boldsymbol{\xi}}=\boldsymbol{f}(t,\boldsymbol{\xi})$. The second motion is formulated with an autonomous DS. For both motions, the unstable estimate of $\boldsymbol{f}(.)$ is learned with four different regression techniques including ϵ-SVR, GMR, GPR, and LWPR. As the estimation of the energy function is independent from the regression method, we use the same energy function to generate the stabilizing command for all the four regression techniques[19]. Furthermore, this feature also allows comparing these four techniques and then choosing the one that fits the task best. In this section we evaluate these approaches based on the stabilizing effort; however, one could also use other criteria depending on the requirements of the task. The stabilizing effort can be measured through the integration of the stabilizing command along trajectories. Given an initial point $\boldsymbol{\xi}^0$, we have:

$$U(\boldsymbol{\xi},\boldsymbol{f}(.);\boldsymbol{\theta})=\sum_{t=0}^{t^f}\|\boldsymbol{u}(\boldsymbol{\xi}^t,\boldsymbol{f}(.);\boldsymbol{\theta})\|\delta t \tag{4.47}$$

where the integration of $\boldsymbol{\xi}^t$ through time is computed according to Eq. (4.4). The lower $U(\boldsymbol{\xi},\boldsymbol{f}(.))$, the more stable behavior $\boldsymbol{f}(.)$ has along that trajectory, and thus the less it is distorted. If $U(\boldsymbol{\xi},\boldsymbol{f}(.))=0$, it means that $\boldsymbol{f}(.)$ is naturally stable for the given initial point. A more intuitive measure of the stabilizing effort can also be obtained by computing the fraction of the total traveled length that these corrections amount to:

$$C.E.=\frac{\sum_{t=0}^{t^f}\|\boldsymbol{u}(\boldsymbol{\xi}^t,\boldsymbol{f}(.);\boldsymbol{\theta})\|\delta t}{\sum_{t=0}^{t^f}\|\dot{\boldsymbol{\xi}}^t\|\delta t}=\frac{\sum_{t=0}^{t^f}\|\boldsymbol{u}(\boldsymbol{\xi}^t,\boldsymbol{f}(.);\boldsymbol{\theta})\|}{\sum_{t=0}^{t^f}\|\dot{\boldsymbol{\xi}}^t\|} \tag{4.48}$$

where C.E. stands for Correction Effort.

The quantitative comparison between these techniques for each motion is summarized in Table 4.1. The results are computed on a set of four and two test trajectories for the first and second motions, respectively. The test data sets are indicated with black squares in Fig. 4.31. In the first motion, the best result in terms of the average stabilizing command is obtained with ϵ-SVR. LWPR also demonstrates a comparative performance to ϵ-SVR for this motion.

[19] The following values are used in this experiment: $\bar{w}=0.0001$, $\kappa_{\boldsymbol{\xi}}=1$, $\sigma_{\boldsymbol{\xi}}=0.001$, $\kappa_{\dot{\boldsymbol{\xi}}}=1$, $\sigma_{\dot{\boldsymbol{\xi}}}=0.0001$, $\tau=20$, and $\delta t=0.01$ second.

Table 4.1: Performance comparison between ϵ-SVR, GMR, GPR, and LWPR in learning the two planar motions of Fig. 4.31. Obtained values are averaged across the demonstrations.

Motion	Criterion	ϵ-SVR	GMR	GPR	LWPR
#1	U (m)	**0.0063**	0.0276	0.0368	0.0072
	C.E.	**1.87%**	7.52%	10.84%	2.20%
#2	U (m)	0.0106	**0.0048**	0.010	0.0148
	C.E.	2.10%	**0.99%**	1.96%	2.99%

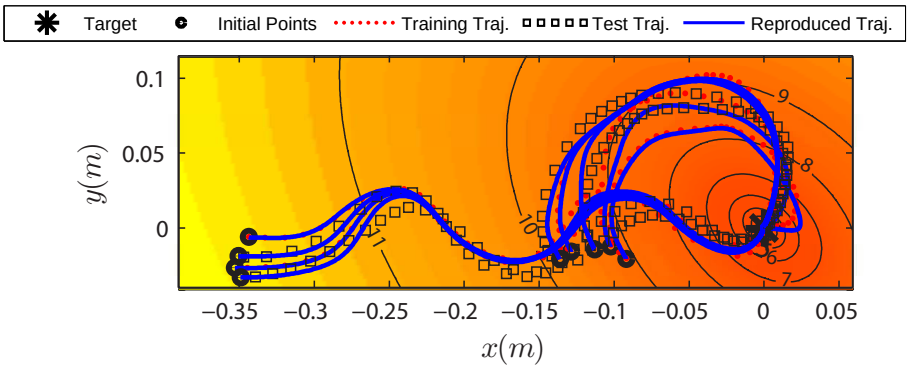
Figure 4.31a shows the obtained trajectories using the ϵ-SVR model. For the second motion, GMR performs much better than the other three methods in terms of the average stabilizing command. The trajectories obtained using the GMR model are shown in Fig. 4.31c.

The norm of stabilizing commands for both motions are shown in Figs. 4.31b and 4.31d, respectively. As we can see, only in small parts of the trajectories, the stabilizing command is active in order to ensure convergence to the target. The small values of the correction effort in Table 4.1 also verifies this observation. For the first and second motions, These values are 1.87% and 0.99%, respectively, which supports our original motivation that the estimates from $\boldsymbol{f}(.)$ mostly exhibit a converging behavior in a region close to demonstrations.

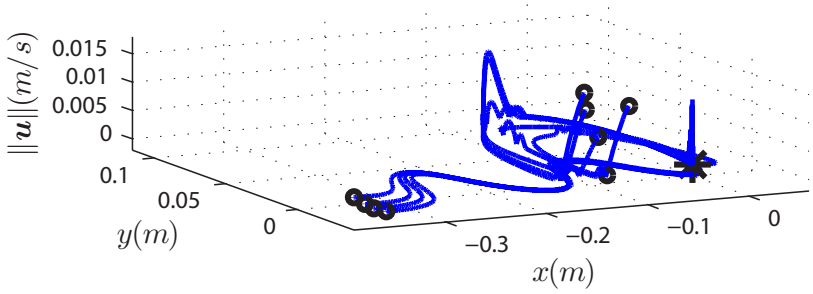
Furthermore, the activation of the stabilizing command on a small region around the target verifies that $\boldsymbol{f}(.)$ is unstable in both motions, and without using the presented approach, all trajectories would have missed the target. Finally, one can observe that in both motions the generated trajectories resemble the user demonstrations despite applying the stabilizing command. This is mainly due to the fact that the stabilizing command is derived from an energy function that is estimated based on the user demonstrations.

The robot experiment consisted of having the WAM arm place an orange on a plate and into a bucket. First, the placing task on the plate is shown to the robot seven times via kinesthetic teaching (see Fig. 4.32a). A DS estimate of this motion is constructed using GMM with 7 Gaussian functions. The energy function is also estimated based on the user demonstrations using the QEF parameterization. Note that for the sake of clarity of Fig. 4.32, the energy contour curves are not illustrated; however, one could imagine them as a set of ellipsoid curves elongated along the $x-y$ plane, and centered at the target.

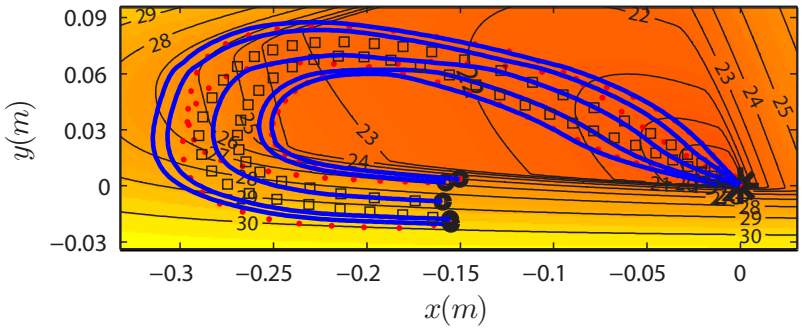
The demonstrations and the reproductions of the task from the proposed method are illustrated in Fig. 4.32b. As it is illustrated, the reproductions closely follow the demonstrations, while their global stability is ensured. Although this model can successfully generate motions to place the orange on the plate, it cannot be used with the bucket (see the solid black lines in Fig. 4.33a). In order to adapt the robot motions to this change without retraining the whole model, we exploit the online learning power of LWPR to locally modify the DS given by GMR:



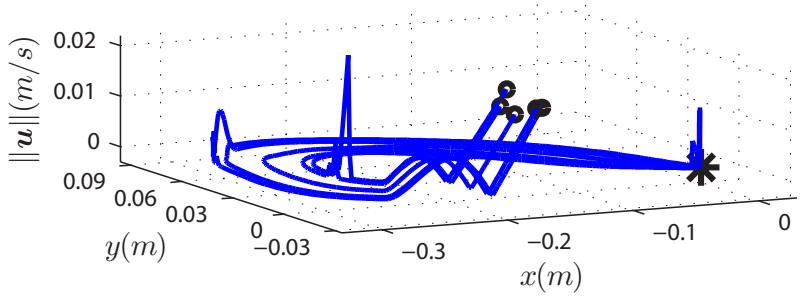
(a) The energy function of the first motion is formulated with the QEF parameterization, and its DS model is estimated with ϵ-SVR.



(b) Illustration of the norm of the stabilizing command for the first motion.



(c) In the second motion, an unstable estimate of the motion is modeled as an autonomous DS using GMR, and its energy function is estimated using the WSAQF parameterization with $\mathscr{L} = 3$.



(d) Illustration of the norm of the stabilizing command for the second motion.

Figure 4.31: Performance evaluation of the proposed algorithm with two planar motions. The background color illustrates the energy level. The lighter the color, the higher the energy.

$$\dot{\boldsymbol{\xi}} = \tilde{\boldsymbol{f}}(\boldsymbol{\xi}) = \underbrace{\boldsymbol{g}(\boldsymbol{\xi}) + \boldsymbol{l}(\boldsymbol{\xi})}_{\boldsymbol{f}(\boldsymbol{\xi})} + \boldsymbol{u}(.) \tag{4.49}$$

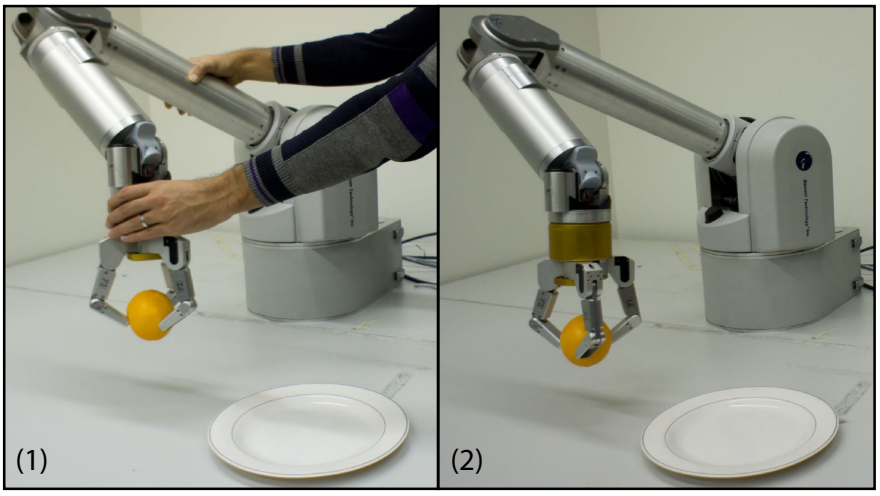
where $\boldsymbol{g}(\boldsymbol{\xi})$ and $\boldsymbol{l}(\boldsymbol{\xi})$ correspond to the DS that are modeled with GMR and LWPR, respectively. Initially $\boldsymbol{l}(\boldsymbol{\xi}) = 0$, $\forall \boldsymbol{\xi} \in \mathbb{R}^d$. The LWPR model is trained incrementally and online by interactively correcting the robot movement while it approaches the bucket (see Fig. 4.33b). The blue hollow circles in Fig. 4.33a shows the new training data points that were collected interactively as we have explained. Figure 4.33c illustrates the reproductions from the combined DS according to Eq. (4.49). The stabilizing command is generated using the same energy function as before. With the new model, the robot can successfully adapt its motion and place the orange into the bucket. Note that in this experiment, the GMR model grants the base behavior for the placing task, and the LWPR model provides the required adaptation to the environment. Anytime when it is necessary, the base behavior can be retrieved by canceling out the LWPR term in Eq. (4.49). By extension, one can also imagine having several LWPR models, each of which provides the required adaptive behavior for different containers.

4.6.6 Discussion and Conclusion

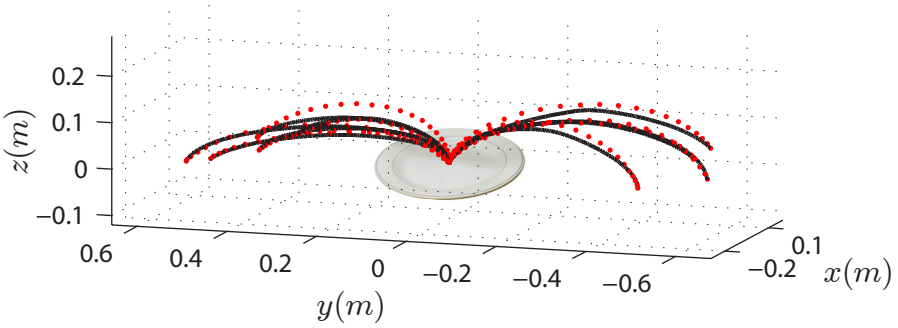
In this section, we presented a new technique, called SEDS-II, to ensure the global asymptotic stability of DS that are estimated from a set of demonstrations. Compared to our previous two approaches for estimation of stable DS, SEDS-II allows using a wide variety of regression techniques, and at the same time it can ensure stability of both autonomous and non-autonomous DS. Moreover, in the light of the proposed framework, it is also possible to merge the advantages of different regression techniques in modeling robot motions, while ensuring the global asymptotic stability of the integrated DS at the target. All these features enable users to choose the most appropriate regression technique(s) for the task at hand, which could result in reaching a higher performance when performing DS-based robot motions.

The presented approach proceeds in two steps: 1) Estimation of a valid energy function, i.e. the so-called metric of stability, from the demonstrations of the task, and 2) Generation of a stabilizing command using the estimated energy function. The former can be estimated by solving a constrained optimization problem, and is an offline procedure. The latter generates the stabilizing command online to correct the motion when a diverging behavior is observed (according to the estimated metric of stability).

To solve the optimization problem, the presented approach requires parameterization of the energy function. The choice of parameterization can influence the accuracy of the final result, hence it should be chosen appropriately. The WSAQF parameterization presented in this section allows to estimate the energy function of complex motions while benefiting from solving a less complex



(a) The demonstration of the task through kinesthetic teaching (left), and its execution by the robot (right).

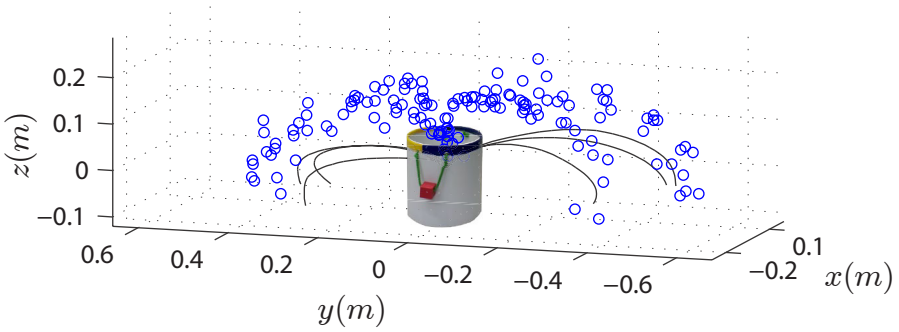


(b) Generation of trajectories from different initial points

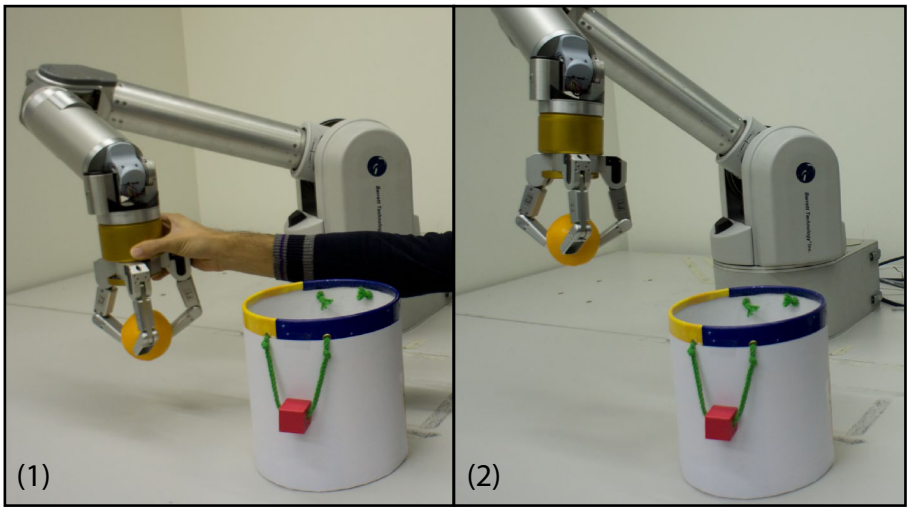
Figure 4.32: The robot experiment of placing an orange on a plate. The placing motion is modeled with GMR.

optimization problem by eliminating the need to mesh the state space to verify the optimization constraints (see Section 4.6.2). Although this feature significantly reduces the training time, it puts an upper bound on the range of energy functions that can be accurately encoded with WSAQF. In fact the WSAQF parameterization can only build an accurate model of the energy function from demonstrations if the following condition holds: "For each demonstration trajectory $\{\boldsymbol{\xi}^t, \dot{\boldsymbol{\xi}}^t\}_{t=0}^T$, if there exit a scalar $m > 1$ and indices $0 \leq i, j \leq T$, $i \neq j$ such that $\boldsymbol{\xi}^{t_i} = m\boldsymbol{\xi}^{t_j}$, then $t_i < t_j$." This condition is derived from the fact that with the WSAQF parameterization, the energy of the system should increase by radially moving away from the target.

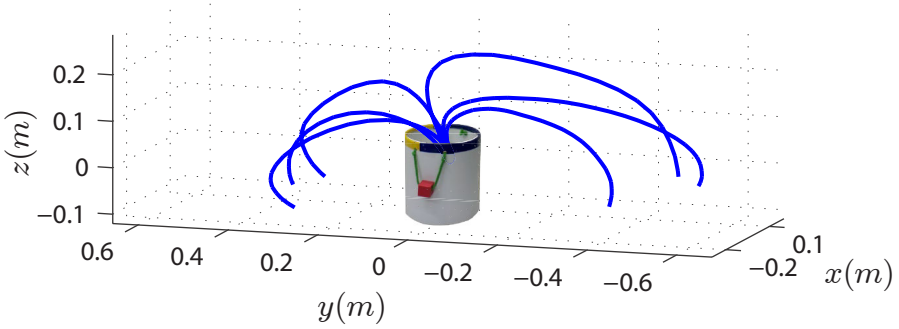
The above criterion simply implies that the demonstrations should not contain an unstable spiral-shape movement (e.g. consider a throwing movement with energy pumping, which requires the amplitude of the swing to increase in order to pump energy to the system). An example of such motions is illustrated in Fig. 4.34. The part of the demonstrations that contains the unstable spi-



(a) By replacing the plate with a bucket, the previous model can no longer be used. To adapt to the new situation, the user interactively modifies the robot trajectory as the robot approaches the bucket. These data are used for online training of the LWPR model. The new training data points for the LWPR model are shown with blue hollow circle. The solid black lines represents trajectories generated with solely using the GMR model.



(b) Training interactively the DS model in order to adapt to the new situation (left). The execution of the reaching motion with the combined LWPR+GMR DS (Right).



(c) Trajectories generated with the combined GMR+LWPR DS. The generated trajectories can successfully place the orange into the bucket.

Figure 4.33: Adaptation of the DS for the case where the plate is replaced with the bucket.

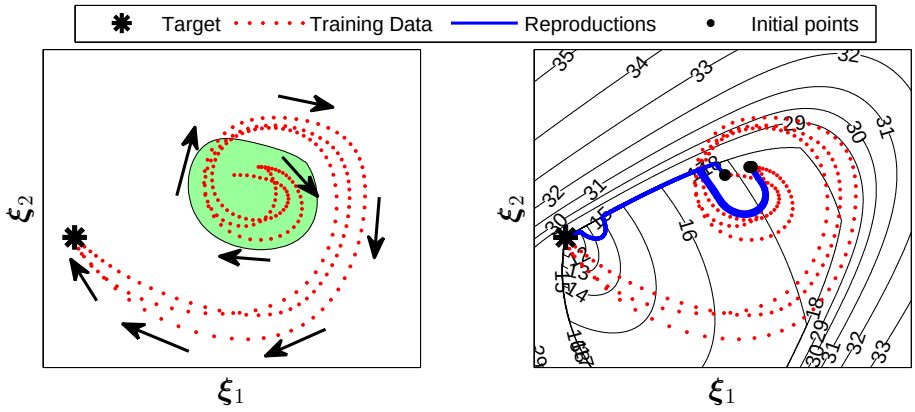


Figure 4.34: An example of a motion that cannot be accurately estimated with SEDS-II. The arrows in the left figure indicates the evolution of the motion. Due to the unstable spiral part of the movement (points inside the green area), the WSAQF parameterization cannot accurately learn the energy function of the motion, and as a result, generated trajectories partly follow the demonstrations.

ral movement is highlighted in green. As it is illustrated, the energy function does not correctly describe the underlying motion, and subsequently the generated trajectories do not follow accurately the demonstrations (note that the inaccurate estimation of the energy function does not compromise the global asymptotic stability of the DS at the target). Remark that this motion cannot be learned with SEDS either.

For such motions, one could decompose the motion into two or more segments and then learn each segment separately. This solution is equivalent to the funnel-based approaches as each funnel takes the motion from one part of the task space and guides it to the basin of attraction of the next funnel until it reaches the target.

4.7 Quantitative Comparison

In this chapter we have presented three methods to build a stable estimate of DS from demonstrations. In this section we aim at providing a more clear understanding of the advantages and limitations of each method, which could eventually help the reader to choose the most suitable approach depending on the task at hand. To reach this goal, we compare the performance of these approaches against each other, and four of the best performing regression methods to date namely GPR, GMR, LWPR, and SVR. We consider two variant of SEDS, i.e. SEDS-likelihood and SEDS-MSE, and we use SEDS-II in combination with the unstable estimates from GPR, GMR, LWPR, and SVR.

The comparison is made on the same library of handwriting motions presented in Section 4.5.2.1, which is composed of 20 planar motions. For each

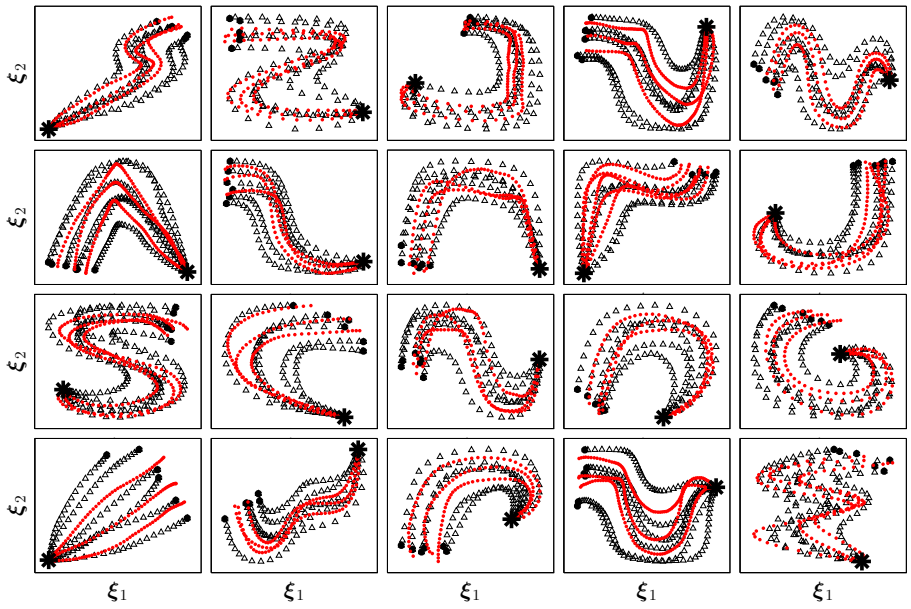


Figure 4.35: Demonstrations of the training and test trajectories for the 20 nonlinear 2D motions that are used for comparison across BM, SEDS, and SEDS-II.

motion, the evaluation is made on a set of six test trajectories that are spread in between and outside the training trajectories (see Fig. 4.35). All reproductions were generated in simulation to exclude the error due to the robot controller from the modeling error. The QEF parametrization is also used to encode the energy function of all the 20 motions in the library.

Quantitative performance comparisons in terms of the accuracy in estimation, the number of required parameters, and the training time are summarized in Fig. 4.36. The qualitative comparison of the estimate of the handwriting motions are provided in Appendix B. We use the swept error area as a metric to evaluate the estimation accuracy of each method (see Eq. (4.28)). The result on the estimation accuracy are shown in Fig. 4.36a. As we can see, all the methods are capable of providing the same order of accuracy in estimations. However, they significantly differ in the number of parameters that they use to model the motions. In terms of the number of parameters, one can clearly see that SEDS-MSE, SEDS-likelihood, and SEDS-II with GMR encoding have significantly fewer number of parameters compared to the other remaining approaches[20] (see Fig. 4.36b). Hence, these three methods are noteworthy in that they provide the same level of accuracy as the other approaches while requiring less parameters. The comparison result on the training time indicates that

[20]GPR's learning parameter is of order of $d(d+1)$; however, it also requires keeping all training datapoints to estimate the output $\dot{\boldsymbol{\xi}}$. Additionally, one needs to store the kernel function $\boldsymbol{K}(\boldsymbol{\Xi}_{\boldsymbol{\xi}}, \boldsymbol{\Xi}_{\boldsymbol{\xi}})$ and its inverse in order to improve the runtime performance of GPR. Hence the total number of required parameters is $d(d+1) + 2dn + 2n^2$, where n is the total number of datapoints. As for the BM, the number of parameters to model the invariant set D is also included in the comparison.

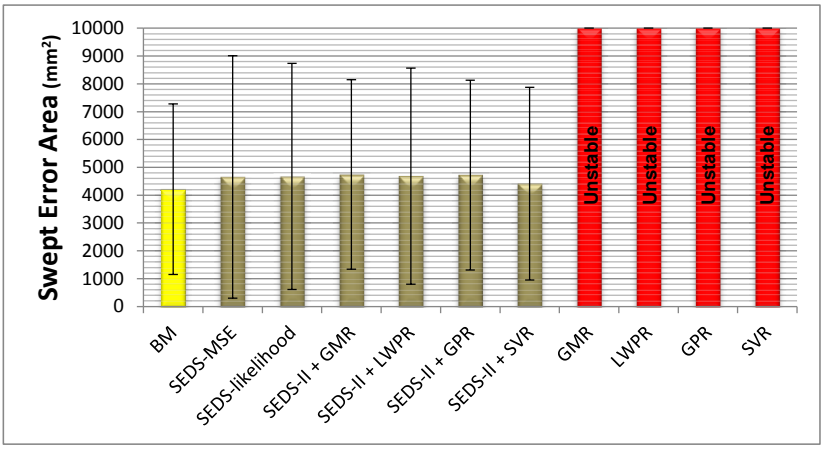
SEDS-II with GMR and SVR formulations are the fastest methods that can build a stable estimate of DS within a few seconds (see Fig. 4.36c). Note that all the methods have a retrieval time of less than a millisecond.

The comparison between GMR, LWPR, GPR, and SVR and their stabilized version through SEDS-II indicates that the addition of the stabilizing command not only makes the system globally stable, but it does that by only requiring 4 additional parameters (which is almost negligible compared to the total number of parameters used by the regression method). The training time to learn the energy function is also small and is about a second.

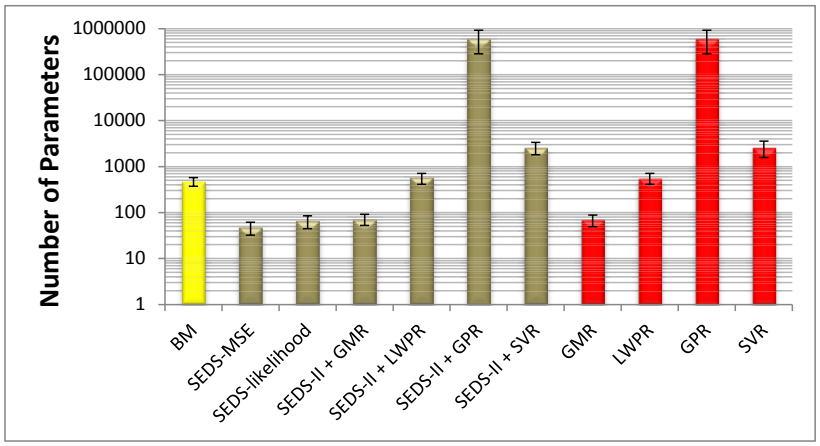
As outlined before, the energy function of all the 20 handwriting motions were encoded using the QEF parameterization. Now let us compare our three approaches on a set of more advanced motions where the use of WSAQF parameterization becomes crucial in order to model the energy function more accurately. We conduct this comparison between BM, SEDS-likelihood, and SEDS-II with GMR encoding on a set of five motions (see Fig. 4.37). Note that for brevity we omit the other possible approaches as the different variants of SEDS and SEDS-II provide the same order of accuracy.

The comparison result is illustrated in Fig. 4.38. As we can see, while both BM and SEDS-II perform equally well, the performance of SEDS is significantly degraded. This is due to fact that the stability conditions in SEDS are derived based on a quadratic energy function. Hence, SEDS cannot model accurately the motions that requires more complex energy function. In other words, for these motions, the stability conditions are more conservative. As it is illustrated in Fig. 4.37, the trajectories generated from the SEDS model deviate from demonstrations in order to respect the conditions derived from the quadratic energy function. In contrast, the trajectories from SEDS-II can follow the demonstrations as it allows using more complex energy functions.

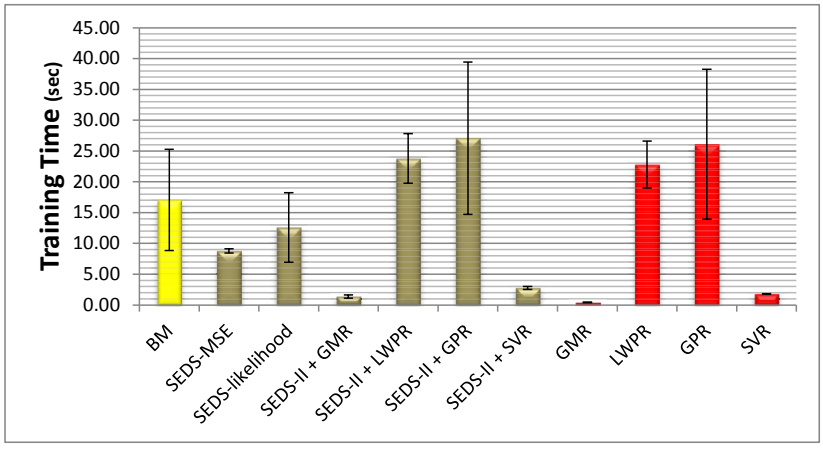
Putting together the results from the above comparison, SEDS-II provides the most flexible yet accurate means of building stable DS from demonstrations. The accuracy of SEDS-II is comparable to BM, yet it ensures global asymptotic stability of DS at the target. In contrast to the SEDS stability conditions, SEDS-II benefits from having less conservative stability conditions which allows learning a wider set of tasks. Moreover, SEDS-II allows using different regression techniques to encode motions, which is advantageous especially when online learning is required. The use of SEDS is preferable if 1) the energy of the motion can be represented with the QEF parameterization, and 2) neither online learning nor other regression techniques than GMR is required for the task. In these cases, SEDS can (marginally) outperforms SEDS-II as it requires fewer number of parameters while it provides the same order of accuracy as SEDS-II. Despite the accuracy of BM, we no longer use this approach in the remaining parts of this thesis as we are mainly interested in tasks that require a large domain of applicability.



(a)

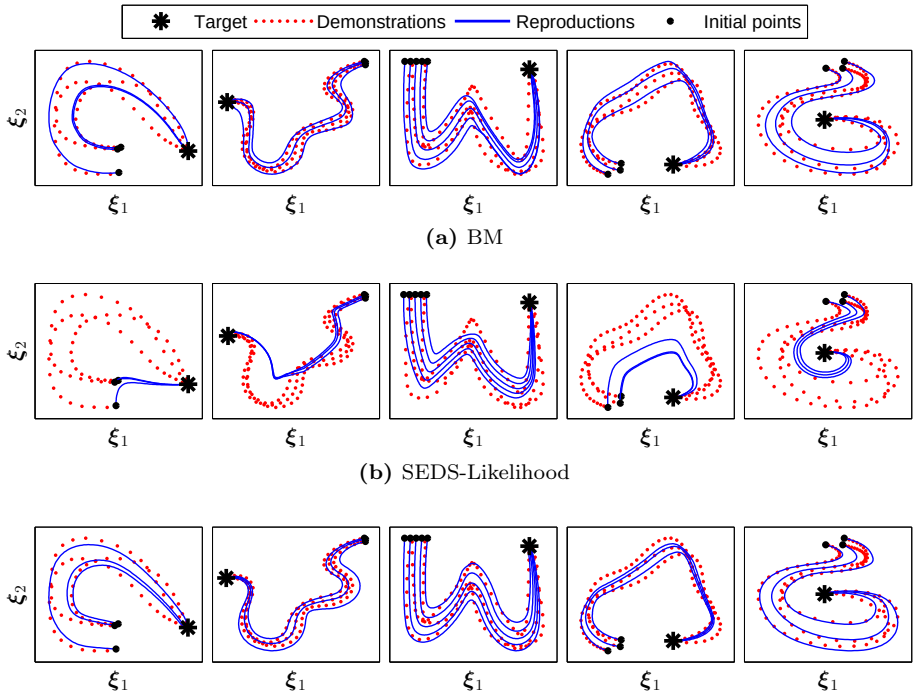


(b)



(c)

Figure 4.36: Performance comparison between the presented approaches and the state of the art regression techniques on a library consists of 20 human handwriting motions. The colors yellow, green, and red respectively indicate whether an approach ensure local stability, global stability, or do not consider stability. For further information and discussion please refer to Section 4.7.



(a) BM

(b) SEDS-Likelihood

(c) SEDS-II with GMR encoding. From left to right, the WSAQF energy function is modeled with $\mathscr{L} = 3$, 1, 1, 2, and 1 asymmetric quadratic functions, respectively.

Figure 4.37: Qualitative comparison between BM, SEDS-likelihood, and SEDS-II with GMR encoding on a set of five advanced motions.

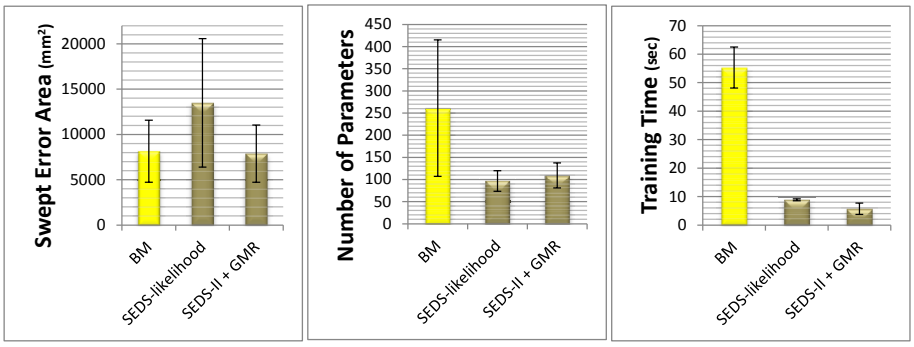


Figure 4.38: Performance comparison between BM, SEDS-likelihood, and SEDS-II with GMR encoding on a set of five advanced motions, where the use of WSAQF parameterization becomes crucial in order to model the energy function more accurately.

4.8 Summary and Conclusion

In this chapter, we presented a DS-based framework to generate robot reaching motions. We addressed the stability problem of such systems, and formulated sufficient conditions to ensure their local and global asymptotic stability at the target. Then, we presented three techniques, namely BM, SEDS, and SEDS-II, to statistically build an estimate of DS-based motions from a set of demonstrations under the strict stability constraint.

We showed that there are two promising advantages inherent in modeling robot motions with globally asymptotically stable DS. 1) Trajectories that are generated from a stable DS are guaranteed to reach the target point (if it is reachable) from any point in the state space, and 2) By modeling motions with DS, one could benefit from an inherent robustness to perturbations and instant adaptation to changes in a dynamic environment. By placing the origin of the reference coordinates system at the target, our DS formulation is invariant with respect to the target position. Furthermore, the learned movements with the proposed approaches are scale invariant. Thus, the speed of movement can be modulated with a positive factor $\kappa > 0$, i.e. $\dot{\boldsymbol{\xi}} = \kappa \boldsymbol{f}(\boldsymbol{\xi})$, without modifying the generated trajectories[21]. This property is essential especially when it is desired to control the timely execution of motions.

From the implementation perspective, as DS encodes an abstract representation of the motion (instead of storing the whole trajectories), it is computationally light and can be used in realtime. In addition to the above features, our SEDS-II approach also supports online/active learning which allows user to refine the model in an interactive manner.

There are a number of structural difference between the presented approaches, which make their application particular to certain tasks. These difference are summarized in Table 4.2. For a given task, one first needs to consider these differences in order to choose the most appropriate approach. For example, if the task is defined in 2D and is solely described in a local region, then BM is the most appropriate method. The BM should be mainly applied to planar motions as it becomes computationally intractable in higher dimensions. If online learning is required, one should go for the SEDS-II formulation, and so on.

In addition to the structural differences, we conducted a quantitative comparison between our presented approaches. The results indicate that for simple movements (i.e. the ones that can be described with a quadratic energy function), BM and all variants of SEDS and SEDS-II provide the same order of accuracy. Hence the choice of model depends mainly on the requirements of the task (e.g. if online learning is required), preference of the user (e.g. familiarity

[21] Theoretically, trajectories generated from $\boldsymbol{f}(\boldsymbol{\xi})$ and $\kappa \boldsymbol{f}(\boldsymbol{\xi})$ exactly coincide on each other, and the only difference between the two systems is in the reaching time to the target. However, in practice, some discrepancies between the two systems may be observed due to the numerical integration error (see Eq. (4.4)).

Table 4.2: Summary of the structural differences between BM, SEDS, and SEDS-II.

Method	**BM**	**SEDS**	**SEDS-II**
Type of stability	local	global	global
Supported DS	autonomous	autonomous	auto./non-auto.
Regression model	GMR	GMR	any
Learning mode	offline	offline	offline/online[†]
Multiple motions encoding	No	Yes	Yes
Computational complexity	Heavy	light	light

[†] The online learning support depends on the type of regression that is selected.

with a particular approach), and also other criteria such as the training time and the number of parameters. Note that as the training is usually done offline and within a few seconds, the training time might not be considered as a decisive factor. However, in terms of the number of parameters, GMR formulation is advantageous as it requires considerably fewer number of parameters. For complex motions, that cannot be described with a quadratic energy function, BM and SEDS-II are superior to SEDS.

An assumption made throughout this chapter is that robot motions can be modeled with a first order ODE. While this type of DS is generic enough to model a wide variety of robot motions, one could expect some special cases where it fails to encode motions properly. Most of the time, this limitation can be tackled through a change of variable. For example a self-intersecting trajectory or a motion for which the starting and final points coincide with each other (e.g. a triangular motion) cannot be modeled through Eq. (4.2) if $\boldsymbol{\xi}$ codes solely for the end-effector cartesian position (i.e. $\boldsymbol{\xi} = \boldsymbol{x} \Rightarrow \dot{\boldsymbol{\xi}} = \dot{\boldsymbol{x}}$). But, if information about velocity is added (i.e. $\boldsymbol{\xi} = [\boldsymbol{x}; \dot{\boldsymbol{x}}] \Rightarrow \dot{\boldsymbol{\xi}} = [\dot{\boldsymbol{x}}; \ddot{\boldsymbol{x}}]$), the system can disambiguate the direction of motion at the intersection and hence successfully encode the dynamics of motion.

The presented approaches control the robot trajectories at the kinematic level. Thus, as outlined before, we assume there is a low level tracking controller[22] that converts kinematic variables into motor commands (see Fig. 4.2). The above control scheme is often associated with one main concern: "the hardware limitations of the robot, such as the torque limit, are not considered at the level of trajectory generation". However, contrary to the classical planer, this concern is not very critical when using a DS-based model for trajectory generation. In fact, the DS model can compensate for deviations (due to hardware limitations) from the desired trajectory, by instantly adapting a new trajectory for the new position of the robot. In other words, it treats the robot's hardware limitations similarly to perturbations. It should be noted that an inevitable outcome of such compensation is that the robot executes the motion at a slower pace than what is expected.

[22] Depending on the platform, we used either a PID (for the Hoap-3, Katana, and iCub robots) or an inverse dynamics controller (for the WAM and DLR arms) to generate motor commands (for further information see Section 2.4).

All theorems derived in this section are based on the continuous state space assumption; however, in real experiments, robot motions are usually generated with a finite number of points (discrete modeling). Thus the choice of integration time step is important as a big integration time step could cause instability in the system even though the continuous DS model is globally asymptotically stable. However, this should not be such an issue as most of the robotic systems usually operate in a sufficiently high frequency (e.g. the WAM arm operates at 500Hz).

As it was pointed out before, there is an upper bound on the range of motions that can be accurately encoded with our presented approaches. BM is mainly applicable to 2D motions. SEDS and SEDS-II can be used to model motions that their energy function can be encoded with QEF and WSAQF parameterizations, respectively. As a result, motions that contain unstable spiral movements are not supported neither by SEDS nor by SEDS-II. For such motions, one could use a funnel-based control technique, where the motion in each funnel can be modeled by a SEDS or SEDS-II model. There are a number of special considerations that should be taken into account when using a funnel-based approach. For example, the funnels should be arranged so that the global asymptotic stability of the system can be ensured (to avoid looping). Although it is an interesting research topic, the generation of funnel-based motions is beyond the scope of this thesis as the majority of the discrete motions we may face in our daily lives can be encoded with the presented approaches.

In contrast to a single trajectory-based imitation learning approach with DMP (see Section 3.3.2), here we take a state space approach to learn the attractor landscape of the entire state space from demonstrated trajectories. Some of the main advantages of our approach over DMP are that 1) it does not need implicit time dependency to ensure stability and thus is more robust to perturbations, 2) it provides a better generalization of the motion since it can shape the target landscape based on several demonstrations, and 3) it is multi-dimensional and thus can capture information about the correlation across different axes (which is essential for accurate modeling of motions).

However, it should be noted that the training time in DMP is single-shot, whereas our approach requires a fair amount of training time (e.g. about a minute in our experiments) to build a stable estimate of DS. Furthermore, although it is theoretically possible to model high-dimensional motions (i.e. $d >> 7$) with our approach, in practice, it is non-trivial to provide sufficient demonstrations to cover such a high-dimensional state space. In contrast, DMP models multi-dimensional systems by learning a separate DS for each dimension and thus does not suffer from the curse of dimensionality (at the cost of introducing inaccuracies in estimation). We will discuss later on in Section 7.2 how our approach can be extended to higher dimensional motions (such as motions on a high degrees of freedom humanoid robot) through introducing coupling terms.

In the next chapters, we will extend our DS-based formulation in two directions. First in the next chapter we suggest a reformulation that allows learning striking movements that requires hitting the target (instead of reaching it) at a specific speed and direction. Then in Chapter 6 we propose a DS-based obstacle avoidance that can instantly modify the robot's motion to avoid collision with multiple static and moving convex obstacles.

Chapter 5

Learning Hitting Movements with Dynamical Systems

> *There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.*
>
> Charles Antony Richard Hoare

In the previous chapter we presented possible mechanisms to build a DS model of reaching motions from a set of user demonstrations. We stressed the importance of ensuring stability of DS in order to provide useful control policies, i.e. generating trajectories that reach the target, and are inherently robust to perturbations. However, there is a limitation inherent to stable DS: a robotic system that is driven by a stable DS can only reach the target with zero-velocity. Hence, it cannot be used to model striking movements that by definition require hitting the target with a *non-zero velocity*.

In this chapter, we provide a reformulation to the DS model presented in Chapter 4, and substitutes the notion of (local or global) stability with (local or global) convergence. This reformulation allows encoding a considerably wider set of tasks ranging from reaching movements to agile robot movements that require hitting a given target with a specific speed and direction. We validate our approach in the context of playing minigolf[1] as a benchmark task.

The goal in minigolf is to sink[2] the ball into a hole located on the field. This task is challenging for humans as it requires precise control for several factors such as orientation, position and speed at the target. There are often obstacles and curved surfaces between the ball and the hole to make the task more complex for the player. Satisfying all these requirements needs many practices.

Studies show that the human players in ball games such as minigolf usually follow the same pattern of motion when approaching the ball, even if circumstances such as the ball position change (Ramanantsoa & Durey, 1994; Mülling

[1] Also commonly referred to as mini-golf, miniature golf, midget golf, and Putt-Putt.
[2] Sinking means hitting the ball such that it goes into the hole.

et al., 2011). To play this game, a player needs first and foremost to learn how to swing the golf club so as to hit the ball precisely. Additionally, depending on the situation, the ball may have to be hit at a particular angle and speed. Human players can achieve this by rotating their body prior to hitting the ball, and adapting the hitting speed. In this work, we follow a similar two-steps approach for teaching minigolf to a robot in which: 1) We first learn the basic motion for hitting the ball, and 2) We provide a means to adapt the default hitting motion to strike the ball at a desired speed and direction.

Note that throughout this chapter we assume that the desired hitting speed and direction are known and given. An extension of this work to learn the hitting parameters was conducted by Klas Kronander, during his master thesis under my advice, and beginning of PhD thesis at LASA. This work was conducted in close collaboration by the two of us and is hence reported in Appendix D to this thesis.

The remainder of this chapter is structured as follows: Section 5.1 gives a formal description of the focus of this chapter. Section 5.2 formalizes the hitting motion, and presents a learning algorithm to estimate it from a set of demonstrations. Section 5.3 provides the final workflow of the complete system, describing both the learning and task execution procedures. Section 5.4 evaluates the performance of the proposed method in robot experiments of playing minigolf on the 6-DoF Katana-T arm, equipped with a golf club tool. Finally, Section 5.5 is devoted to the conclusion. Unless otherwise specified, throughout this chapter we represent motions in Cartesian coordinates system, i.e. $\boldsymbol{\xi} = \boldsymbol{x} = [x\ y\ z]^T$.

Note that the work presented here are published as two joint papers with Klas Kronander in (Khansari-Zadeh et al., 2012; Kronander et al., 2011). This chapter reports solely on the parts that were developed by myself. The results that were obtained collaboratively are provided in Appendix D.

Related publications:

- S.M. Khansari Zadeh, K. Kronander, and A. Billard (2012), Learning to Play Minigolf: A Dynamical System-based Approach, *Advanced Robotics*, p. 1–27.

- K. Kronander, S.M. Khansari Zadeh, and A. Billard (2011), Learning to Control Planar Hitting Motions in a Minigolf-like Task, *In proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 710–717, *Received the JTSC Novel Technology Paper Award*.

- S.M. Khansari Zadeh and A. Billard (2011), Learning to Play Mini-Golf from Human Demonstration using Autonomous Dynamical Systems, *In electronic proceedings of the Workshop on New Developments in Imitation Learning*, International Conference on Machine Learning (ICML).

5.1 Problem Statement

In the minigolf task considered in this thesis, the player only gets one chance to sink the ball. To achieve this, first of all, the player must know how to swing the golf club to hit the ball. This requires having general knowledge about how to hit the ball in various situations depending on the position of the player, the ball, and the hole. For example consider Fig. 5.1a. In this example, there are several initial positions where the robot is required to start a swing motion and hit the ball along a desired direction. This is a non-trivial task, which cannot be simply fulfilled by just playing recorded trajectories. Additionally, playing in dynamic environments where the ball or the hole could be displaced during the swing phase requires an online and smooth adaptation of the swing motion in order to fulfill the task and to sink the ball. Similarly to Chapter 4, we consider a DS approach to model the hitting motion. When encoding the hitting motion with an autonomous DS, this problem reduces to estimating a smooth first order differential equation $\boldsymbol{f}_h(\boldsymbol{\xi})$:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}_h(\boldsymbol{\xi}) \qquad \boldsymbol{f}_h : \mathbb{R}^3 \mapsto \mathbb{R}^3 \tag{5.1}$$

The main challenge in estimating $\boldsymbol{f}_h(\boldsymbol{\xi})$ is to ensure that starting from any initial point $\boldsymbol{\xi}^0 \in \mathbb{R}^3$, the temporal evolution of the motion passes through the target point (i.e. hits the ball) at a desired speed and direction, while retaining the main features presented in demonstrations.

Now assume the player has learned a planar hitting motion and can hit the ball in a direction specified by the unit vector $\boldsymbol{\psi}^* \in \mathbb{R}^2$ in the hitting plane and with hitting speed $v^* \in \mathbb{R}^+$. For each new situation, a hitting angle α and a hitting speed gain κ must be chosen such that hitting with speed κv^* in direction $\boldsymbol{\psi}^*_\alpha = \boldsymbol{R}_\alpha \boldsymbol{\psi}^*$ (where R denotes a counterclockwise rotation by α in the hitting plane) leads to sinking the ball (see Fig. 5.1b). Provided that the hitting angle and speed are known, it would be advantageous if the DS $\boldsymbol{f}_h(\boldsymbol{\xi})$ can be adapted so that it hits the ball with the desired hitting parameters. We will address the problem of learning the hitting motions and its adaptation to different situations in Section 5.2.

Aside from learning the hitting motions, estimating the hitting parameters is also an interesting question and a potentially very hard task for advanced fields. Consider the simplest possible minigolf field: a flat field without obstacles. Such a field is depicted in Fig. 5.1b. In this case the choice of hitting angle is trivial - the ball should simply be hit in a straight line towards the hole. The vector $\boldsymbol{s} \in \mathbb{R}^2$ denotes the relative position of the hole to the ball projected in the hitting plane. This vector represents the *situation* that the player has to adapt to when choosing the hitting parameters. As can be seen in Fig. 5.1b, to play the flat field, the player simply has to align the hitting direction $\boldsymbol{\psi}^*_\alpha$ with this vector. With the correct hitting angle, the player can use a wide range of speeds that result in sinking the ball.

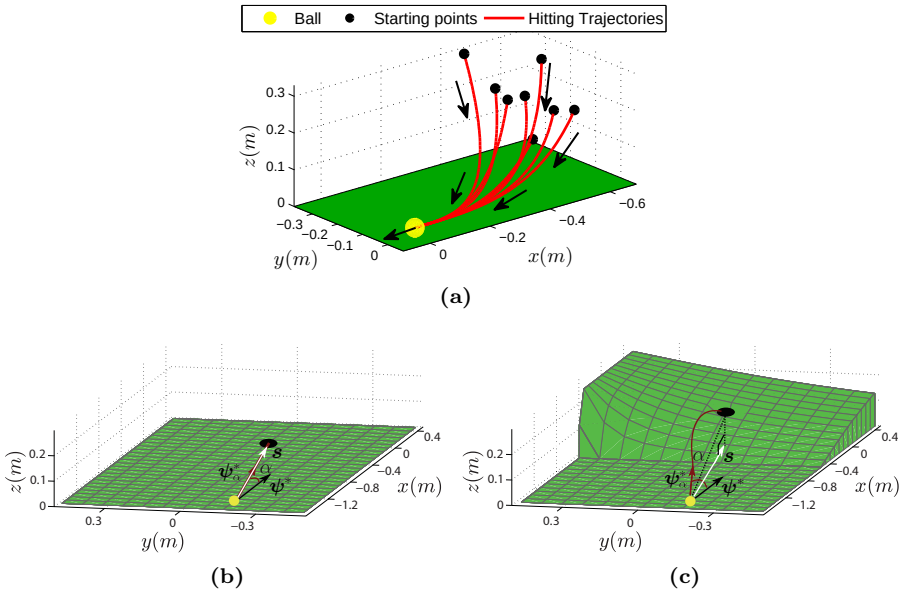


Figure 5.1: Illustration of challenges in playing mini-golf. **(a):** For mastering in minigolf, the player needs to know how to swing the golf club to hit the ball in various situations. **(b) & (c):** Situation on a flat and an advanced fields. The ball trajectory of a successful attempt is indicated by the red line. For the flat field, the hitting direction should be aligned with the input vector $\boldsymbol{s}$. For the advanced field, a larger hitting angle must be chosen so as to compensate for the slope of the field.

Now consider the more advanced field such as the arctan field[3] (see Fig. 5.1c). The vector $\boldsymbol{s}$, describing the situation, is identical in both figures. If the player chooses to hit the ball along $\boldsymbol{s}$ as on the flat field, the ball will not be sunk. To compensate for the slope, a hitting angle larger than the one used for the flat field must be chosen, resulting in a curved trajectory of the ball. Changing $\boldsymbol{s}$ means that a new angle and speed must be selected accordingly. Thus, the player needs to be able to estimate the hitting angle α and hitting speed gain κ given the situation on the field $\boldsymbol{s}$.

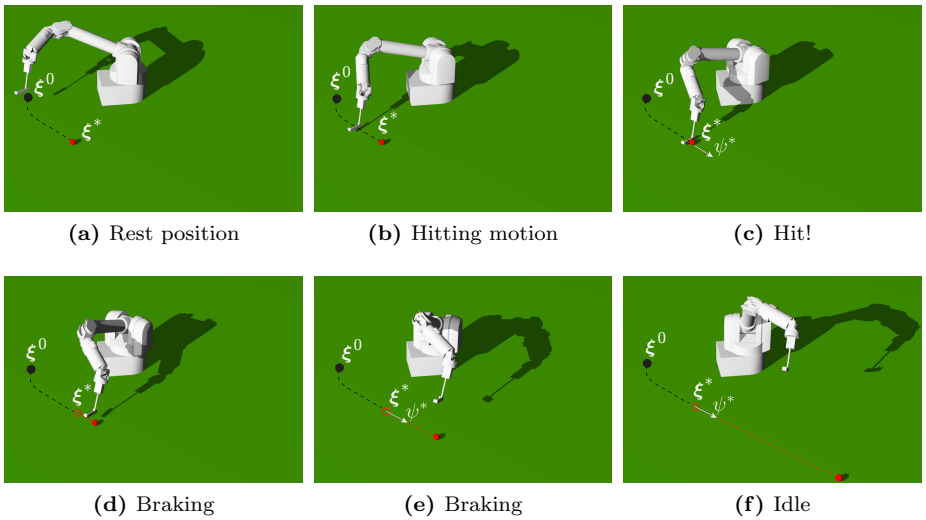
$$\boldsymbol{g} : \boldsymbol{s} \mapsto (\alpha, \kappa) \tag{5.2}$$

where $\boldsymbol{g}$ is a function providing the mapping between the hitting parameters and the situation on the field. This problem is covered in Appendix D.

5.2 Hitting Motions

As outlined in the previous section, one of the requirements for the minigolf task is a default hitting motion. The hitting motion must be flexible so that the hitting direction and the hitting speed can be changed without relearning the whole motion pattern. Learning a hitting motion is similar to point-to-point

[3]The shape is a scaled evaluation of the arctan function over a grid.



(a) Rest position **(b)** Hitting motion **(c)** Hit!

(d) Braking **(e)** Braking **(f)** Idle

Figure 5.2: The 7-DoF WAM arm at different stages of the hitting motion. **(a)** The robot is in its rest position, awaiting to initialize the hitting motion. **(b)** The hitting motion has been started and the robot accelerates the end-effector towards the ball. When the end-effector reaches the hitting point in **(c)**, the DS motion is switched to a braking mode. **(d)** and **(e)** The robot is in the braking mode, gently lifting the golf club while smoothly decelerating the joints. **(f)** The robot goes into idle mode when all the joints have stopped moving.

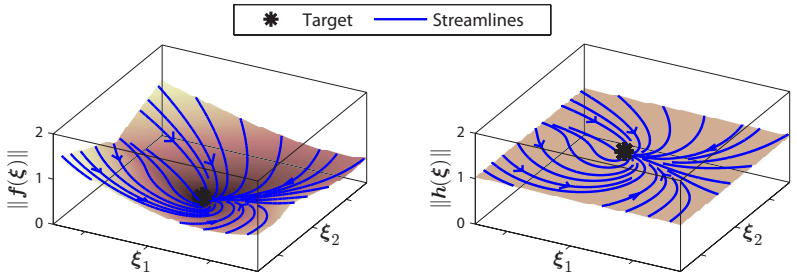
motions which was studied in the previous chapter, with the difference that instead of ensuring global asymptotic stability of DS motion at the target, we now need to guarantee the convergence of all trajectories to the target.

In this section we propose an extension to our DS formulation to support the above change. The structure of our formulation is analogous to many physical principles where the motion of a particle in space can be defined with the value of a vector field (e.g. gravity, electrical field, etc.) times a scalar (e.g. mass, electric charge, etc.). Using this analogy, we formalize robot motions with a *target field* $\boldsymbol{h}(\boldsymbol{\xi})$ and a *strength factor* $v(\boldsymbol{\xi})$:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}_h(\boldsymbol{\xi}) = v(\boldsymbol{\xi})\boldsymbol{h}(\boldsymbol{\xi}) \tag{5.3}$$

The target field defines for each point $\boldsymbol{\xi}$ in the taskspace a normalized vector specifying the direction of motion, and the strength factor is a scalar indicating the speed with which the robot should move along the specified direction.

Figure 5.2 illustrates an example of the minigolf hitting motion. Here, the robot starts the motion from an initial point $\boldsymbol{\xi}^0$, and approaches the ball located at position $\boldsymbol{\xi}^*$, aligning with the hitting direction $\boldsymbol{\psi}^*$ before hitting the ball. In this example, the target field represents different techniques employed by the player to hit the ball (e.g. flop and chip shots in golf or topspin and slice hits in tennis), and the strength factor is a modulation factor to produce different desired hitting speeds.



(a) Streamlines of $\boldsymbol{f}(\boldsymbol{\xi})$ **(b)** Streamlines of $\boldsymbol{h}(\boldsymbol{\xi})$

Figure 5.3: Comparison of streamlines of a globally stable model $\boldsymbol{f}(\boldsymbol{\xi})$ with the target field $\boldsymbol{h}(\boldsymbol{\xi})$. Though both functions have exactly the same streamlines, the value of $\boldsymbol{h}(\boldsymbol{\xi})$ does not vanish while approaching the target.

5.2.1 Target Field

To achieve our goal of having a target field that produces trajectories that always pass through the target point with a *non-zero* velocity, we reformulate our DS modeling presented in Chapter 4. To combine the property of a stable DS with the possibility to reach the target with a non-zero velocity, we define the target field as a normalized flow of motion induced by a globally stable dynamics $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$:

$$\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta}) = \frac{\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})}{\|\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})\|} \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^3 \backslash \boldsymbol{\xi}^* \tag{5.4}$$

Equation (5.4) corresponds to a field with constant intensity (i.e. $\|\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})\| = 1$), and is defined anywhere except the target, where $\|\boldsymbol{f}(\boldsymbol{\xi}^*;\boldsymbol{\theta})\| = 0$. To overcome the singularity at the target point, in practice, we use the value of the target field in the previous time step instead of evaluating $\boldsymbol{h}(\boldsymbol{\xi}^*;\boldsymbol{\theta})$. The flow induced by $\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})$ is of constant speed. In contrast, the flow generated from $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$ varies based on the current state of the motion defined by $\boldsymbol{\xi}$. The vector field $\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})$ conserves the convergence properties at the attractor of $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$ and follows strictly the same streamlines (see Fig. 5.3).

Considering Eq. (5.4), the problem of estimating the target field $\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})$ reduces to finding a globally stable DS $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$. All three methods presented in Chapter 4 could in principle be used to build an estimate of the target field. However, these methods are not ideally suited for this task since the estimate of $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$ tries to reproduce not just the direction but also the speed of movement. As the amplitude of $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$ does not affect the estimation of the target field $\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})$, the error in estimating $\|\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})\|$ should not be penalized. Therefore, in this section we suggest a slightly modified version of SEDS that solely penalizes for the error in estimating the direction of the movement. This can be achieved by using the inner product as a means to quantify the accuracy of estimations based on the demonstrations[4].

[4]The same type of modification can be applied to BM and SEDS-II models. For brevity, we do not provide such modification here as all the experiments considered throughout this chapter can be accurately modeled with the SEDS model.

Optimization problem: Given a set of N demonstrations $\{\boldsymbol{\xi}^{t,n}, \dot{\boldsymbol{\xi}}^{t,n}\}_{t=0,n=1}^{T^n,N}$, the optimal value of the unknown parameters $\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}^1..\boldsymbol{\mu}^K; \boldsymbol{\Sigma}^1..\boldsymbol{\Sigma}^K\}$ of the function $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$ are obtained by solving:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\sum_{t=0}^{T^n} \omega^{t,n} \frac{\left(\dot{\boldsymbol{\xi}}^{t,n}\right)^T \boldsymbol{f}(\boldsymbol{\xi}^{t,n};\boldsymbol{\theta})}{\|\dot{\boldsymbol{\xi}}^{t,n}\|\|\boldsymbol{f}(\boldsymbol{\xi}^{t,n};\boldsymbol{\theta})\|} \tag{5.5}$$

subject to

$$\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}} = \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1}(\boldsymbol{\mu}^k_{\boldsymbol{\xi}} - \boldsymbol{\xi}^*) \qquad \forall k \in 1..K \tag{5.6a}$$

$$\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} + (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1}(\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}})^T \prec 0 \qquad \forall k \in 1..K \tag{5.6b}$$

$$\boldsymbol{\Sigma}^k \succ 0 \qquad \forall k \in 1..K \tag{5.6c}$$

$$0 < \pi^k \leq 1 \qquad \forall k \in 1..K \tag{5.6d}$$

$$\sum_{k=1}^{K} \pi^k = 1 \tag{5.6e}$$

where $(.)^T$ denotes the transpose, $. \prec 0$ corresponds to the negative definiteness of a matrix, and $\boldsymbol{f}(\boldsymbol{\xi}^{t,n};\boldsymbol{\theta})$ are computed directly from Eq. (4.8). The optimization constraints given by Eq. (5.6) ensure the global asymptotic stability of $\boldsymbol{f}(\boldsymbol{\xi};\boldsymbol{\theta})$, and by extension the global convergence of all trajectories driven by $\boldsymbol{h}(\boldsymbol{\xi};\boldsymbol{\theta})$ to the target (see Section 4.5.1). The positive weighting factors $\omega^{t,n}$ determine the relative importance of each point when computing the estimated error. In this work, we give a lower weight ω_l and an upper weight ω_u to the initial and final points of each demonstration, respectively. For intermediary points, the weighting factors are computed by linearly interpolating between these two values:

$$\omega^{t,n} = \frac{t}{T^n}(\omega_u - \omega_l) + \omega_l \tag{5.7}$$

Thus the optimization tries to fit the last parts of the movement better, when the effect of deviation from a desired trajectory becomes more important. Throughout this chapter we use an interior-point algorithm to solve this optimization problem (for further information about this algorithm refer to Section 2.3).

5.2.2 Strength Factor

In order to be able to generate robot motions with similar velocity profiles as the demonstrations, we use the strength factor to modulate the target field according to Eq. (5.3). The strength factor v is a *positive scalar*, and defines the intensity of the motion that the robot should follow. Note that the global convergence at the hitting point is guaranteed by the target field alone. This offers flexibility as the speed profile can be changed independently of the target field. To capture nonlinearities in the velocity profile, we consider a varying strength factor that depends on the state variable, i.e. $v(\boldsymbol{\xi}) : \mathbb{R}^d \rightarrow \mathbb{R}^+$.

An estimate of the strength factor $v(\boldsymbol{\xi})$ can be learned from the same demonstrations as those for the target field using various existing regression techniques. In this work, we use GMR; however, one can expect similar results using other techniques[5]. As outlined in Section 2.2.1, the parameters of the Gaussian Mixture Model are optimized through Expectation Maximization (EM) (Dempster & Rubin, 1977). EM finds an optimal model of $v(\boldsymbol{\xi})$ by maximizing the likelihood that the complete model represents the data well. Using GMR, the strength factor is thus given by:

$$v(\boldsymbol{\xi}) = \sum_{k=1}^{K_{SF}} h_{SF}^k(\boldsymbol{\xi}) \left(\boldsymbol{\Sigma}_{SF,v\boldsymbol{\xi}}^k (\boldsymbol{\Sigma}_{SF,\boldsymbol{\xi}}^k)^{-1} (\boldsymbol{\xi} - \boldsymbol{\mu}_{SF,\boldsymbol{\xi}}^k) + \boldsymbol{\mu}_{SF,v}^k\right) \tag{5.8}$$

where π_{SF}^k, $\boldsymbol{\mu}_{SF}^k$, and $\boldsymbol{\Sigma}_{SF}^k$ are priors, means and covariance matrices of component k in the GMM model of the strength factor. The nonlinear weighting $h_{SF}^k(\boldsymbol{\xi})$ is computed in the same way as described by Eq. (4.9). The subscript SF for Strength Factor is used above to clarify that two different GMMs are involved in the reproduction of the hitting motion.

5.2.3 Control of Hitting Direction

Equation (5.3) provides the trajectory dynamics of the end-effector with the hitting speed $v^* \in \mathbb{R}^+$ given by the strength factor and the hitting direction $\boldsymbol{\psi}^* \in \mathbb{R}^3$ defined by the target field at the hitting point.

Thus, the default hitting speed and direction are given during the demonstrations. To adapt these parameters to new situations on the field, we proceed as follows:

1. Hitting in a different direction can be seen as a rotation of the coordinate frame in which the default DS is defined. If α is the angle between the desired and the default hitting directions in the plane of the golf field, the first step is to transform the input to the desired reference frame via the rotation matrix $\boldsymbol{R}_\alpha^T$.
2. The output of the DS needs to be transformed back to our desired hitting direction. Therefore, we rotate back through $\boldsymbol{R}_\alpha$.
3. Finally, the hitting speed can be changed by modulating the DS by some positive gain $\kappa > 0$.

In brief, the following DS models a hitting motion in direction $\boldsymbol{\psi}_\alpha^*$ and with speed κv^*:

$$\dot{\boldsymbol{\xi}} = \kappa\, \boldsymbol{R}_\alpha\, \boldsymbol{f}_h(R_\alpha^T \boldsymbol{\xi}; \boldsymbol{\theta}) = \kappa\, \boldsymbol{R}_\alpha\, v(\boldsymbol{R}_\alpha^T \boldsymbol{\xi})\, \boldsymbol{h}(\boldsymbol{R}_\alpha^T \boldsymbol{\xi}; \boldsymbol{\theta}) \tag{5.9}$$

Figure 5.4 shows an example of using the rotation matrix to control the hitting direction at the target. In this illustration, the default hitting direction $\boldsymbol{\psi}^*$

[5] Note that irrespective of which method is used, the strength factor should not affect the direction of the motion. To ensure this, the constraint $v(\boldsymbol{x}) > 0$ should be taken into account either during learning or regression (e.g. by setting $v(\boldsymbol{x}) = \|v(\boldsymbol{x})\| + \epsilon$)

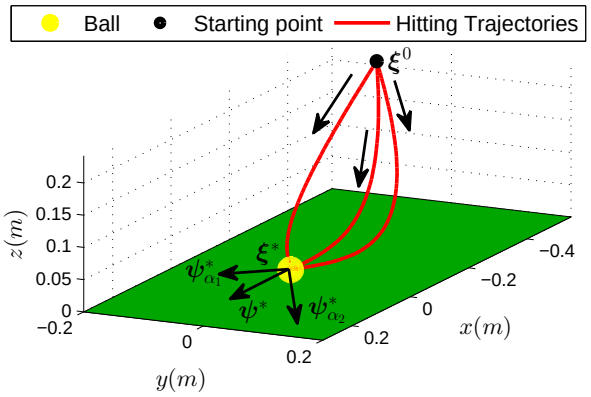


Figure 5.4: Control of hitting direction by using the rotation scheme. The default hitting direction $\boldsymbol{\psi}^*$ is rotated with angles α_1 and α_2 to generate hitting motions with the direction $\boldsymbol{\psi}^*_{\alpha_1}$ and $\boldsymbol{\psi}^*_{\alpha_2}$, respectively.

is rotated with angles α_1 and α_2 to generate hitting motions with the direction $\boldsymbol{\psi}^*_{\alpha_1}$ and $\boldsymbol{\psi}^*_{\alpha_2}$, respectively.

The effect of the impact between the golf club and the ball depends on two things: the Cartesian trajectory of the golf club before hitting the ball (the direction of approach) and the orientation of the golf club at the hitting point. Human players normally align the direction of hitting with the direction of approach by keeping the golf club perpendicular to the direction of approach. In this work, we take the same approach and control the end-effector orientation so as to keep the golf club perpendicularly aligned to the direction of motion when hitting.

Note that for the continuous version of DS, as presented in Eq. (5.3), we can safely go ahead and multiply the whole strength factor with a constant κ without adventuring stability or altering the trajectory. However, when we move to the time discrete domain, altering the speed in this way will change the trajectory depending on the sampling time. In the experiments we present in this chapter, we work with a sufficiently high frequency to allow modulation with a constant speed gain without any noticeable deviation from the trajectory.

5.3 Minigolf Workflow

A conceptual workflow describing the learning and playing of minigolf is given in Fig. 5.5. The left side of this workflow explains the training parts. Three nonlinear models are learned based on two sets of user demonstrations. This procedure is performed offline. The demonstrations of the hitting motions are collected through kinesthetic teaching. With these trajectories, models of the target field and the strength factor are built. These models are used to generate the hitting motion. Training data set of the hitting parameters are then collected

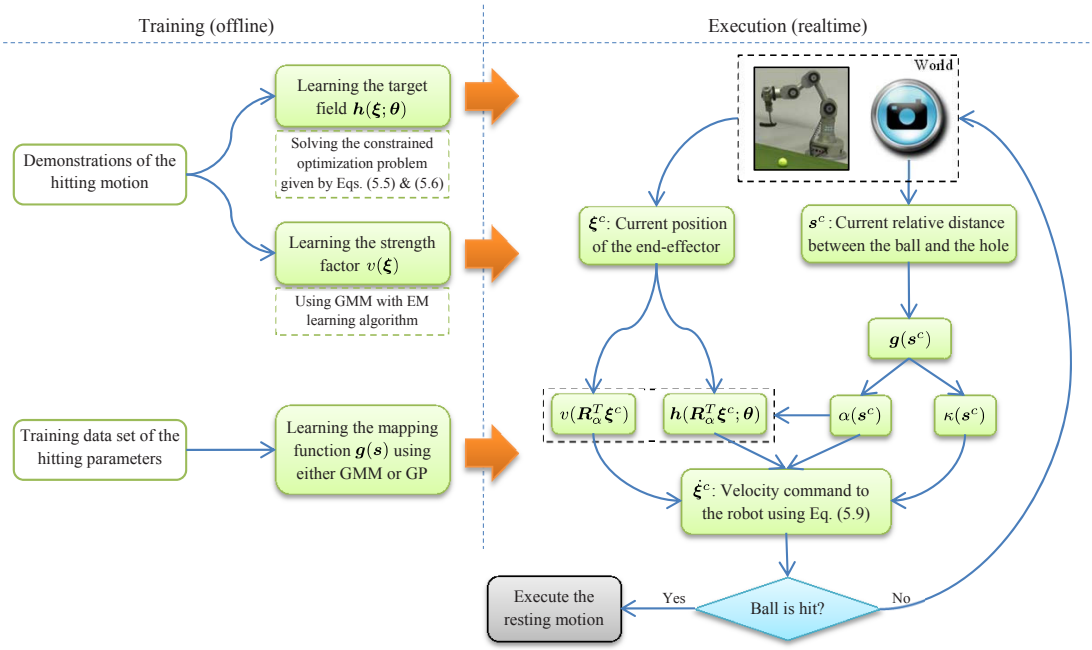


Figure 5.5: A conceptual workflow describing how to play minigolf. For further information please refer to Section 5.3.

by using the hitting motion model, with hitting parameters specified by the teacher. The teacher thus finds some examples of good hitting parameters for some situations, and adds those to the hitting parameters adaptation data set. This data set is then used to estimate a mapping from the situation to the hitting parameters (please refer to Appendix D for further information about how this mapping can be obtained). Only once this hitting parameters adaptation model has been learned, the robot can play minigolf autonomously.

The right side of Fig. 5.5 describes the execution procedure. At each iteration, the current position of the ball, the hole, and the robot's end-effector is updated from the sensors. The correct hitting parameters are then computed using the relative position of the ball to the hole (for further information on the prediction of the hitting parameters, please refer to Appendix D). Based on the value of the hitting angle, the rotation matrix is determined. The target field and the strength factor are then computed using the current position of the robot's end-effector and the rotation matrix. Putting together all these values, the commanded velocity to the robot is calculated using Eq. (5.9). This velocity is then commanded to the robot. The algorithm iterates through the steps above until it hits the ball. When the ball is hit, the motion dynamics are switched to a resting mode to smoothly stop the robot.

5.4 Experimental Results

We evaluated the performance of the presented method in playing minigolf on a flat field using the 6-DoF Katana-T arm. When playing on a smooth flat field, learning the hitting parameters is unnecessary since the hitting direction is aligned with the vector connecting the centers of the ball and the hole (see Fig. 5.1b). The hitting speed can also be preset to a fixed value[6]. The experiments on playing minigolf on challenging fields are reported in Appendix D.

For this experiment, we collected a set of demonstrations by passively moving the robot arm to strike the ball (see Fig. 5.6a). For all demonstrations, the relative position of the ball and the hole was fixed, and the user only showed the robot different ways of hitting the ball starting from different initial positions. In total, seven successful demonstrations were collected and used to learn the task (see Fig. 5.6b). In each demonstration, we recorded the robot's joints angles at 20Hz by directly reading them from each joint's encoder. Forward kinematics was used to compute the Cartesian position and velocity of the end-effector. This data was then used to model the task (i.e. $\boldsymbol{\xi} = [x\ y\ z]^T$ and $\dot{\boldsymbol{\xi}} = [\dot{x}\ \dot{y}\ \dot{z}]^T$). The location of the ball was detected at the rate of 80 fps using two high-speed Mikrotron MK-1311 cameras.

We solved the optimization problem presented in Section 5.2.1 to learn the target field of the hitting motion using $K = 3$ Gaussian functions. This number was selected manually based on a tradeoff between the model's accuracy and the number of parameters needed to encode the motion using the method described in Section 4.5.1. Figure 5.6c illustrates the reproductions of the motion using the final optimized model we obtained from the proposed extended version of SEDS. The strength factor was learned using EM algorithm with 2 Gaussian functions. Figures 5.6d to 5.6f represents the velocity profile of the reproductions versus demonstrations along the axes x, y, and z respectively.

Figure 5.6g shows the sequence of the motion for one of the reproductions. As described in Section 5.2.3, the end-effector orientation was controlled to keep the golf club perpendicular to the direction of approach. At each point, the robot's joint angles were computed by solving the damped least squares pseudo-inverse kinematics with five constraints (3 and 2 constraints for the end-effector's position and orientation, respectively). For each reproduction, after hitting the ball, the dynamics were switched to a stable dynamics guiding the arm into a resting position. For this experiment, we considered a simple resting motion where the velocity of the arm's end-effector gradually decreases along the direction of the motion until it stops.

[6]Recall that a wide range of hitting speeds can be used to sink the ball in the flat field. In this paper, we set a hitting speed of 1 m/s for the experiments on the flat field.

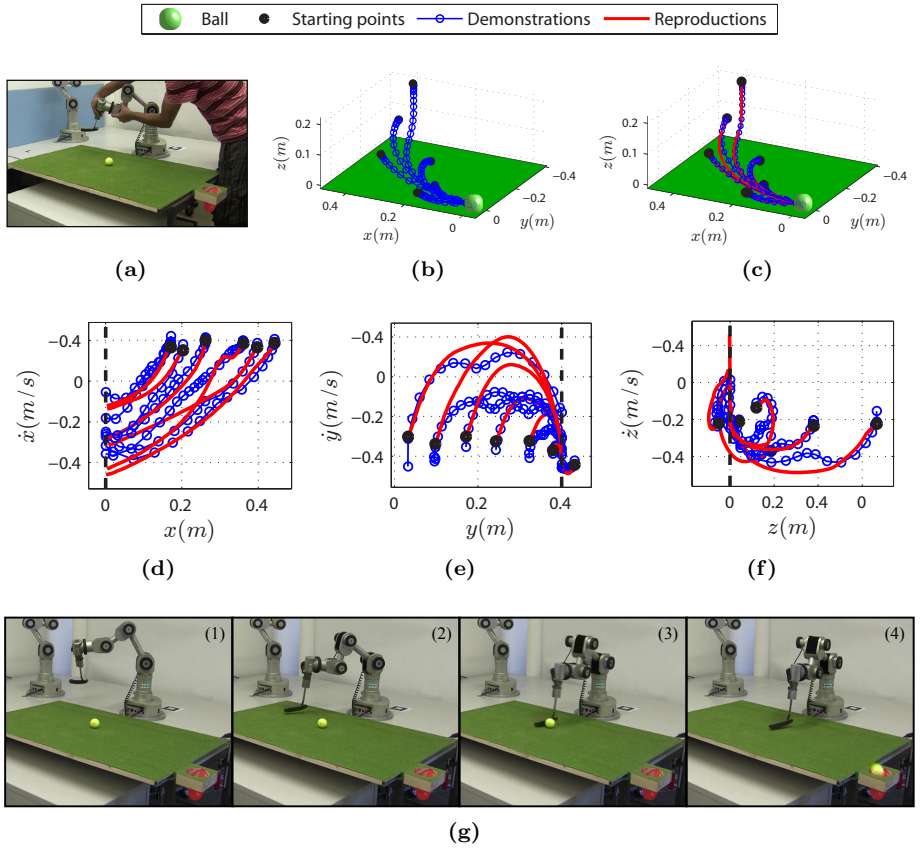
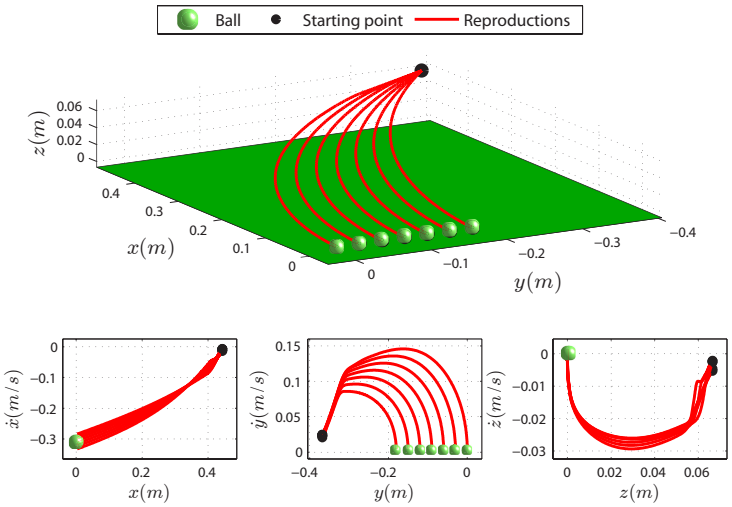


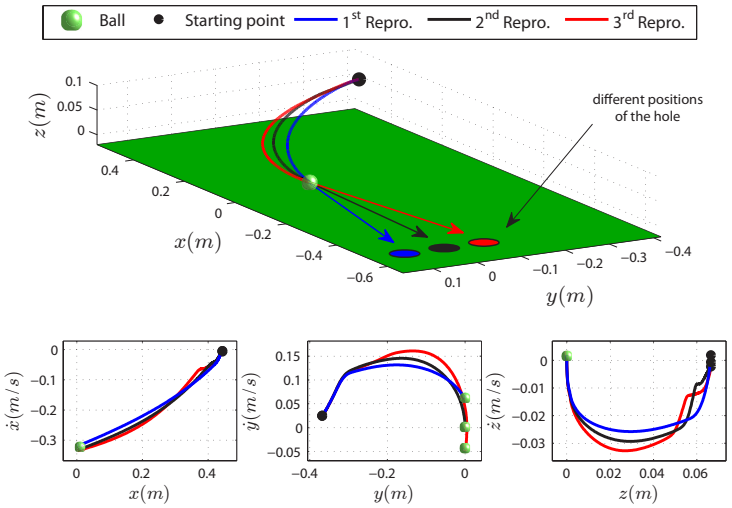
Figure 5.6: **(a)** Kinesthetic demonstration of the putting motion to the 6-DoF Katana-T robot. **(b)** Illustration of the collected successful demonstrations. **(c)** Reproductions of the motion from the model learned with the extended version of SEDS. **(d)-(f)** Evaluation of the model's accuracy in estimating the desired velocity profile. The thick dashed lines locate the position of the ball. **(g)** Illustration of one of the generated motions sequences.

Generalization Ability: Figure 5.7 illustrates the generalization ability of the model to different positions of the golf ball and the hole. In Fig. 5.7a, we changed the position of the ball along the y-axis from 0 to -0.18 m. We also changed the position of the hole so that the vector connecting the center of the ball and the hole always remained along the x-axis. In all cases, the robot successfully hit the ball with the correct speed at the target. Figure 5.7b demonstrates the adaptation of the robot motion to three different positions of the hole. Though the initial configuration of the robot's arm and the ball positions were fixed, the robot took three different paths to hit the ball in the correct direction.

Adaptation to Changes in Dynamic Environments: Similar to all autonomous DS, the proposed model is inherently robust to external perturbations and can provide instant adaptation to changes in the environment. Figures 5.8a and 5.8b illustrate the model's behavior in a dynamic environment. In these



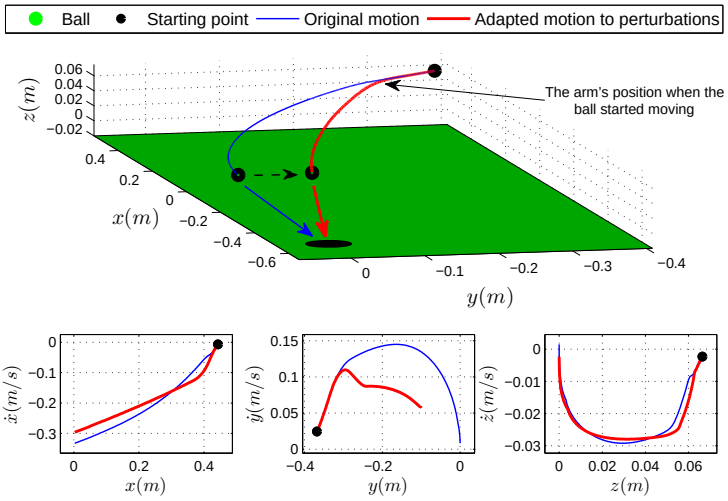
(a) Generalization to different ball positions.



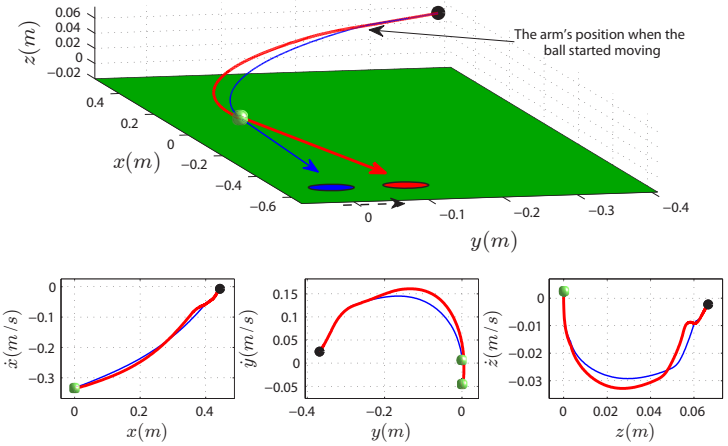
(b) Generalization to different hole positions.

Figure 5.7: Evaluation of the model's performance in generalization.

experiments, during the robot's arm movement, the ball (Fig. 5.8a) and the hole (Fig. 5.8b) were displaced along the negative direction of the y-axis. At each time step, the robot successfully adapted its trajectory to the new position of the ball/hole until it hit the ball. In both examples, the robot successfully managed to hit the ball in the correct direction as the adaptation to the perturbation was done on-the-fly, i.e. without any re-planning. Note that despite the inherent robustness of stable autonomous DS to perturbations, there is an upper bound for the maximum value of perturbations that can be handled. This upper bound is due to the robot's hardware limitations, which affect the maximum acceleration and velocity that the robot can achieve. Thus, if the robot faces a large perturbation when it is close to the ball, it might not be able to react swiftly and hit the ball with the correct hitting direction and speed.



(a) Robustness to changes in the ball position.



(b) Robustness to changes in the hole position.

Figure 5.8: Performance evaluation of the model in a dynamic environment. In this example the ball **(a)** and the hole **(b)** are pushed along the negative direction of the y-axis, as the robot approaches.

5.5 Summary and Conclusion

In this chapter, we extended our previous formulation of DS to generating robot motions with a desired velocity at the target. The new formulation has a similar structure to many physical principles, in that it computes the output of a nonlinear time-independent DS by multiplying the target field with a strength factor. For each point in space the target field indicates the correct direction of the motion, while the strength factor defines the speed of the movement in that direction. Hence it enables a robot to perform motions with similar forms but with different speeds at the target.

Similarly to a globally asymptotically stable DS, the proposed formulation is able to adapt on-the-fly a new trajectory in the face of perturbations without

any need to re-index, re-scale, or re-plan. This is a critical property especially for performing agile motions. For example, in tennis, at the beginning of the motion the estimation of the ball's position is not accurate, but as the ball approaches the robot, this estimation becomes more and more accurate and thus the robot should be able to continuously adapt its motion to the new position of the ball. Note that despite the inherent robustness of stable autonomous DS to perturbations, there is an upper bound for the maximum value of perturbations that can be handled. If, for instance, the robot faces a large perturbation when it is close to the ball, due to the robot's hardware limitations the robot might not be able to react swiftly and hit the ball with the correct hitting direction and speed.

Note that the presented approach is not restricted to playing minigolf, and can be used to generate hitting motions in other tasks such as playing billiard, bowling, etc. These games may require additional hitting parameters such as spin and/or the height of release of the ball to be learned. It should be noted that our approach does not explicitly consider timely execution of the movement as it encodes hitting motions with an autonomous DS. Thus, in tasks where timing becomes crucial (for example in tennis), it relies on an external mechanisms to control the whole motion duration by actively modulating the strength factor, see e.g. the method developed in (S. Kim et al., 2010).

Finally the proposed formulation can be used to define hitting motions in both Cartesian and joint spaces. In this work we have only used the former since it was easier to work with in the context of playing minigolf. However, depending on the task at hand one can choose either of these coordinates systems.

Chapter 6

Dynamical System-based Obstacle Avoidance

Obstacles are those frightful things you see when you take your eyes off your goal.

Henry Ford (1863-1947)

The previous chapters have been devoted to obtain a generic framework that leverages on the notion of DS to generate robot motions that are inherently robust to perturbations and can instantly adapt to changes in the target's position. Despite these features, our framework still relies on a simplistic assumption that presumes there is no object in the robot working space. Such assumption could be very limiting, especially if we envision to bring robots in our daily lives. Many real world tasks require robotic systems that should work in cluttered environments where the robot may face several objects during the task execution. In such environments, it is very likely that the robot may collide with some of these objects and the task would fail. Hence, it is crucial to endow our DS control policy with the collision avoidance capability.

In this chapter, we propose an obstacle avoidance algorithm that can be integrated into our existing DS-based control law, while retaining the swiftness and robustness provided by these approaches. In the presented method, we assume that the robot motion is driven by a continuous and differentiable DS in the absence of obstacle(s). This DS is provided by the user, and henceforth we will call it the *original DS*. Given the original DS and an analytical formulation describing the surface of obstacles, we present an algorithm that can instantly modify the robot's trajectory to avoid collisions with obstacles. This presented work was published in (Khansari-Zadeh & Billard, 2012), and all the material from this publications is collected here.

The rest of this chapter is structured as follows: Section 6.1 formalizes our obstacle avoidance algorithm for robot motions in the presence of a convex obstacle. Section 6.2 discusses the stability of the control law after applying the proposed obstacle avoidance algorithm. Sections 6.3 to 6.5 provide a set of extensions to endow our approach with extra features such as customizing avoidance trajectories, obstacle avoidance in the presence of multiple static/moving obstacles, etc. Section 6.6 gives a conceptual sketch on how to use the proposed algorithm in robot experiments. Section 6.7 presents the experimental results, and Section 6.8 concludes the chapter.

Related publication:

- S.M. Khansari Zadeh and A. Billard (2012), A Dynamical System Approach to Realtime Obstacle Avoidance, *Autonomous Robots*, 32(4), p. 433–454.
- S.M. Khansari Zadeh and A. Billard (2012), Realtime Avoidance of Fast Moving Objects: A Dynamical System-based Approach, *In electronic proceedings of the Workshop on Robot Motion Planning: Online, Reactive, and in Realtime*, The IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).

6.1 Obstacle Avoidance Formulation

Following our general formulation given by Eq. (4.31), we consider robot motions that are defined by an autonomous or non-autonomous DS in the absence of obstacle(s). In this section we show how we can induce a modulation on our generic motion due to the presence of an obstacle. We first consider a hyper-sphere obstacle. We then extend this model to convex objects.

6.1.1 Hyper-Sphere Obstacles

Consider a d-dimensional hyper-sphere object with radius r^o centered at $\boldsymbol{\xi}^o$. The object creates a modulation throughout the robot's state space, which is conveyed through the nonlinear function $\boldsymbol{\phi}^s(\boldsymbol{\xi};\boldsymbol{\xi}^o,r^o):\mathbb{R}^d\mapsto\mathbb{R}^d$ as follows[1]:

$$\boldsymbol{\phi}^s(\boldsymbol{\xi};\boldsymbol{\xi}^o,r^o)=(1+\frac{(r^o)^2}{(\boldsymbol{\xi}-\boldsymbol{\xi}^o)^T(\boldsymbol{\xi}-\boldsymbol{\xi}^o)})(\boldsymbol{\xi}-\boldsymbol{\xi}^o) \tag{6.1}$$

where $(.)^T$ denotes the transpose. To determine how $\boldsymbol{\phi}$ modulates the velocity of the robot, we compute the Jacobian which yields:

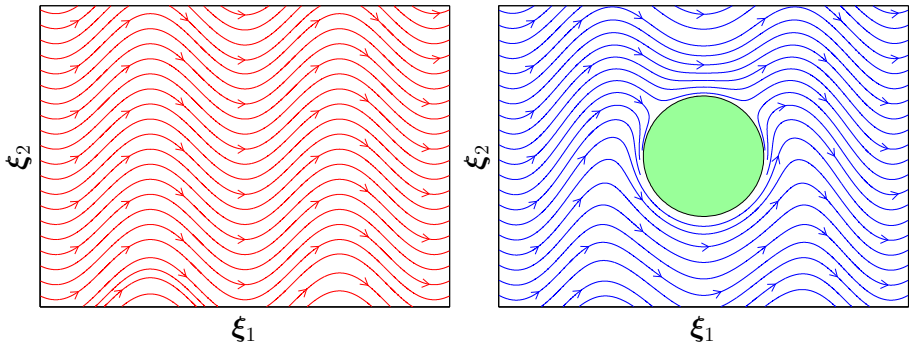
$$\boldsymbol{M}^s(\boldsymbol{\xi};\boldsymbol{\xi}^o,r^o)=\nabla\boldsymbol{\phi}^s(\boldsymbol{\xi};\boldsymbol{\xi}^o,r^o) \tag{6.2}$$

To simplify the notation, we express the modulation in a frame of reference centered on the object and define $\tilde{\boldsymbol{\xi}}=\boldsymbol{\xi}-\boldsymbol{\xi}^o$:

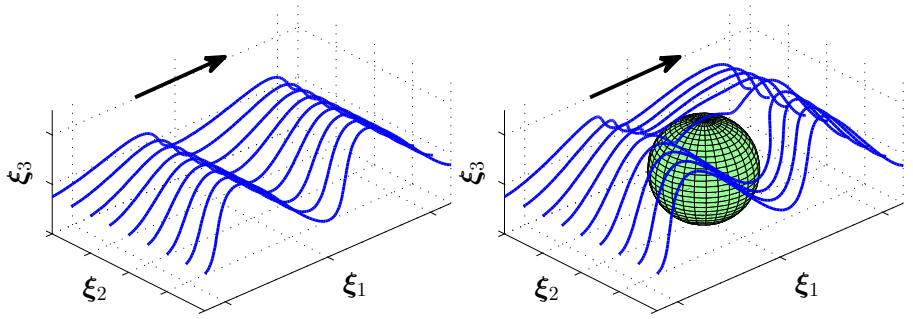
$$\boldsymbol{M}^s(\tilde{\boldsymbol{\xi}};r^o)=\boldsymbol{I}+(\frac{r^o}{\tilde{\boldsymbol{\xi}}^T\tilde{\boldsymbol{\xi}}})^2(\tilde{\boldsymbol{\xi}}^T\tilde{\boldsymbol{\xi}}\boldsymbol{I}-2\tilde{\boldsymbol{\xi}}\tilde{\boldsymbol{\xi}}^T) \tag{6.3}$$

where $\boldsymbol{I}$ is the identity matrix. We call $\boldsymbol{M}^s$ the dynamic modulation matrix. The final model for real-time avoidance of spherical obstacles can be obtained by applying the dynamic modulation matrix to the original DS given by Eq. (4.31):

[1] The development of Eq. (6.1) was partly inspired by the complex potential function that models the uniform flow around a circular cylinder (Milne-Thomson, 1960). In both formulations the modulation of the flow due to the object's presence decreases quadratically with the distance to the center of the object (see the second term in Eq. (6.1)). The main difference between the two approaches lies in their functionality. Equation (6.1) is a d-dimensional vector and its Jacobian is a $d\times d$ matrix which can be used to modulate the original flow. In contrast, the complex potential function is a scalar value, and its derivative directly gives the modified flow in the presence of the obstacle.



(a) Two dimensional flow generated by $\dot{\xi}_1 = 1.0$ and $\dot{\xi}_2 = \sin(\xi_1)$.



(b) Three dimensional flow generated by $\dot{\xi}_1 = 1.0$, $\dot{\xi}_2 = -\sin(\xi_2/4)\sin\xi_1$, and $\dot{\xi}_3 = \sin\xi_1$.

Figure 6.1: Effect of the modulation induced by a spherical obstacle (located at the origin and with radius $r^o = 2$) on a two and three dimensional flows.

$$\dot{\boldsymbol{\xi}} = \boldsymbol{M}^s(\tilde{\boldsymbol{\xi}}; r^o)\boldsymbol{f}(.) \tag{6.4}$$

$\boldsymbol{M}^s(\tilde{\boldsymbol{\xi}}; r^o)$ in Eq. (6.4) is a modulation factor that locally deforms the original dynamics $\boldsymbol{f}(.)$ such that the robot does not hit the obstacle (recall that we use the notation $\boldsymbol{f}(.)$ to refer to both autonomous and non-autonomous DS).

□

Theorem 6.1 *Consider a d-dimensional static hyper-sphere obstacle in* $\mathbb{R}^d$ *with center* $\boldsymbol{\xi}^o$ *and radius* r^o*. The obstacle boundary consists of the hyper-surface* $\boldsymbol{\mathcal{X}}^b \subset \mathbb{R}^d = \{\boldsymbol{\xi} \in \mathbb{R}^d : \|\boldsymbol{\xi} - \boldsymbol{\xi}^o\| = r^o\}$*. Any motion* $\boldsymbol{\xi}(t)$*,* $t \geq 0$ *that starts outside the obstacle, i.e.* $\|\boldsymbol{\xi}(0) - \boldsymbol{\xi}^o\| \geq r^o$*, and evolves according to Eq. (6.4) will never penetrate into the obstacle, i.e.* $\|\boldsymbol{\xi}(t) - \boldsymbol{\xi}^o\| \geq r^o$*,* $\forall t > 0$*.*

Proof: See Appendix A.4.

■

Figure 6.1 illustrates the effect of the modulation induced by such a spherical object on two and three-dimensional flows. As it is illustrated, in both cases the flow is deflected properly and it passes the obstacle.

6.1.2 Convex Obstacles

Suppose a continuous function $\Gamma(\tilde{\boldsymbol{\xi}})$ that projects $\mathbb{R}^d$ into $\mathbb{R}$. The function $\Gamma(\tilde{\boldsymbol{\xi}})$ has continuous first order partial derivatives (i.e. $\mathcal{C}^1$ smoothness) and increases monotonically with $\|\tilde{\boldsymbol{\xi}}\|$. The level curves of Γ (i.e. $\Gamma(\tilde{\boldsymbol{\xi}}) = c$, $\forall c \in \mathbb{R}^+$) enclose a convex region. By construction, the level curve at the surface of the obstacle has value 1:

$$\Gamma(\tilde{\boldsymbol{\xi}}) = 1 \tag{6.5}$$

For example $\Gamma(\tilde{\boldsymbol{\xi}}) : \sum_{i=1}^{d} (\tilde{\boldsymbol{\xi}}_i / \boldsymbol{a}_i)^2 = 1$ corresponds to a d-dimensional ellipsoid with axis lengths $\boldsymbol{a}_i$. Recall that we use the notation $\tilde{\boldsymbol{\xi}}$ to refer to the coordinates of a point $\boldsymbol{\xi}$ in the frame of reference of the obstacle:

$$\tilde{\boldsymbol{\xi}} = \boldsymbol{R}^T (\boldsymbol{\xi} - \boldsymbol{\xi}^o) \tag{6.6}$$

where $\boldsymbol{R}$ is a rotation matrix transforming the obstacle coordinates system to the world coordinates system. We can divide the space spanned by Γ into three regions $\boldsymbol{\mathcal{X}}^o$, $\boldsymbol{\mathcal{X}}^b$, and $\boldsymbol{\mathcal{X}}^f$ to distinguish between points inside the obstacle, at its boundary, and outside the obstacle respectively:

$$\text{Interior points :} \qquad \boldsymbol{\mathcal{X}}^o = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) < 1\} \tag{6.7}$$

$$\text{Boundary points :} \qquad \boldsymbol{\mathcal{X}}^b = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) = 1\} \tag{6.8}$$

$$\text{Free region :} \qquad \boldsymbol{\mathcal{X}}^f = \{\boldsymbol{\xi} \in \mathbb{R}^d : \Gamma(\tilde{\boldsymbol{\xi}}) > 1\} \tag{6.9}$$

At each point $\boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b$ on the outer surface of the obstacle, we can compute a tangential hyper-plane defined by its normal vector $\boldsymbol{n}(\tilde{\boldsymbol{\xi}}^b)$:

$$\boldsymbol{n}(\tilde{\boldsymbol{\xi}}^b) = \begin{bmatrix} \frac{\partial \Gamma(\tilde{\boldsymbol{\xi}}^b)}{\partial \boldsymbol{\xi}_1^b} & \cdots & \frac{\partial \Gamma(\tilde{\boldsymbol{\xi}}^b)}{\partial \boldsymbol{\xi}_d^b} \end{bmatrix}^T \tag{6.10}$$

By extension, we can compute a *deflection hyperplane* at each point $\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}^f$ outside the obstacle with normal:

$$\boldsymbol{n}(\tilde{\boldsymbol{\xi}}) = \begin{bmatrix} \frac{\partial \Gamma(\tilde{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}_1} & \cdots & \frac{\partial \Gamma(\tilde{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}_d} \end{bmatrix}^T \tag{6.11}$$

Each point on the deflection hyper-plane can be expressed as a linear combination of a set of $(d-1)$ linearly independent vectors. These vectors form a basis of the deflection hyper-plane. One particular set of such vectors $\boldsymbol{e}^1, ..., \boldsymbol{e}^{d-1}$ is[2]:

$$\boldsymbol{e}_j^i(\tilde{\boldsymbol{\xi}}) = \begin{cases} -\frac{\partial \Gamma(\tilde{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}_i} & j = 1 \\ \frac{\partial \Gamma(\tilde{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}_1} & j = i \neq 1 \\ 0 & j \neq 1,\ j \neq i \end{cases} \qquad i \in 1..d-1\ ,\ j \in 1..d \tag{6.12}$$

where $\boldsymbol{e}_j^i$ corresponds to the j-th component of the i-th basis vector. Figure 6.2

[2] In case $\partial \Gamma(\tilde{\boldsymbol{\xi}}) / \partial \boldsymbol{\xi}_1$ vanishes, the vectors are no longer linearly independent and one should choose another index for the derivative which is non-zero.

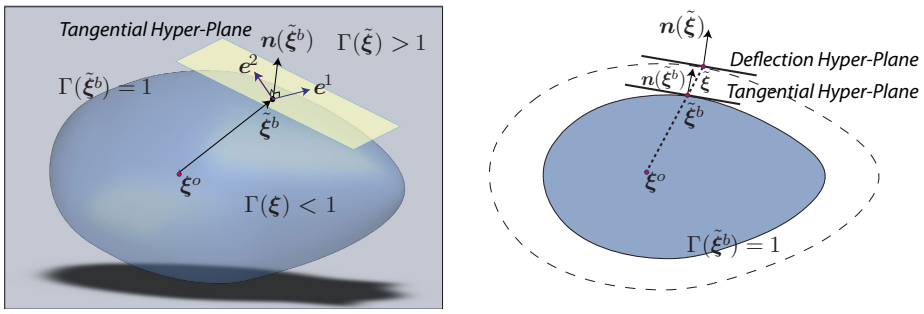


Figure 6.2: Illustration of the tangential hyper-plane and its basis (**left**), and the deflection hyper-plane (**right**) for a 3-dimensional object.

illustrates the tangential and the deflection hyper-planes for a three-dimensional object. We can determine a modulation matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ given by[3]:

$$\boldsymbol{M}(\tilde{\boldsymbol{\xi}}) = \boldsymbol{R}\boldsymbol{E}(\tilde{\boldsymbol{\xi}})\, \boldsymbol{D}(\tilde{\boldsymbol{\xi}})\, \boldsymbol{E}(\tilde{\boldsymbol{\xi}})^{(-1)} \boldsymbol{R}^T \tag{6.13}$$

with the matrices of basis vectors $\boldsymbol{E}(\tilde{\boldsymbol{\xi}})$ and associated eigenvalues $\boldsymbol{D}(\tilde{\boldsymbol{\xi}})$:

$$\boldsymbol{E}(\tilde{\boldsymbol{\xi}}) = \begin{bmatrix} \boldsymbol{n}(\tilde{\boldsymbol{\xi}}) & \boldsymbol{e}^1(\tilde{\boldsymbol{\xi}}) & \cdots & \boldsymbol{e}^{d-1}(\tilde{\boldsymbol{\xi}}) \end{bmatrix} \tag{6.14}$$

$$\boldsymbol{D}(\tilde{\boldsymbol{\xi}}) = \begin{bmatrix} \lambda^1(\tilde{\boldsymbol{\xi}}) & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \lambda^d(\tilde{\boldsymbol{\xi}}) \end{bmatrix} \tag{6.15}$$

where

$$\begin{cases} \lambda^1(\tilde{\boldsymbol{\xi}}) = 1 - \frac{1}{|\Gamma(\tilde{\boldsymbol{\xi}})|} \\ \lambda^i(\tilde{\boldsymbol{\xi}}) = 1 + \frac{1}{|\Gamma(\tilde{\boldsymbol{\xi}})|} \qquad 2 \leq i \leq d \end{cases} \tag{6.16}$$

The dynamic modulation matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ propagates the influence of the obstacle on the motion flow. The result of Eq. (6.13) is invariant to the choice of the basis $\boldsymbol{e}^1..\boldsymbol{e}^{d-1}$. Furthermore, the matrix of basis vector is invertible in $\mathbb{R}^d \setminus \boldsymbol{\xi}^o$. At the obstacle reference point $\boldsymbol{\xi}^o$, the deflection hyper-plane is undefined; however, this does not cause any problem since $\boldsymbol{\xi}^o$ is a point inside the obstacle (recall $\Gamma(\boldsymbol{0}) < 1$). Moreover, since $\Gamma(\tilde{\boldsymbol{\xi}})$ monotonically increases with $\|\tilde{\boldsymbol{\xi}}\|$, the matrix of eigenvalues and by extension the dynamic modulation matrix converge to the identity matrix as the distance to the obstacle increases. Hence, the effect of the dynamic modulation matrix is maximum at the boundaries of the obstacle, and vanishes for points far from it.

Similarly to the hyper-sphere obstacle avoidance given by Eq. (6.4), we can apply the modulation given by Eq. (6.13) on our original motion flow $\boldsymbol{f}(.)$ which yields:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{M}(\tilde{\boldsymbol{\xi}})\boldsymbol{f}(.) \tag{6.17}$$

[3]The derivation of Eq. (6.13) is inspired from the proof of Theorem 6.1. For a spherical obstacle, these equations yield to the same result given by Eq. (6.3).

□

Theorem 6.2 *Consider a convex manifold* $\Gamma(\tilde{\boldsymbol{\xi}}) = 1$ *that encloses a static d-dimensional obstacle with respect to a reference point* $\boldsymbol{\xi}^o$ *inside the obstacle. A motion* $\boldsymbol{\xi}(t)$*, that starts outside the obstacle, i.e.* $\Gamma\big(\tilde{\boldsymbol{\xi}}(0)\big) \geq 1$*, and evolves according to Eq. (6.17) does not penetrate the obstacle, i.e.* $\Gamma\big(\tilde{\boldsymbol{\xi}}(t)\big) \geq 1$, $\forall t > 0$.

Proof: See Appendix A.5.

■

Fig. 6.3 illustrates with four examples the effect of the modulation induced on the field of motion in the presence of different obstacles.

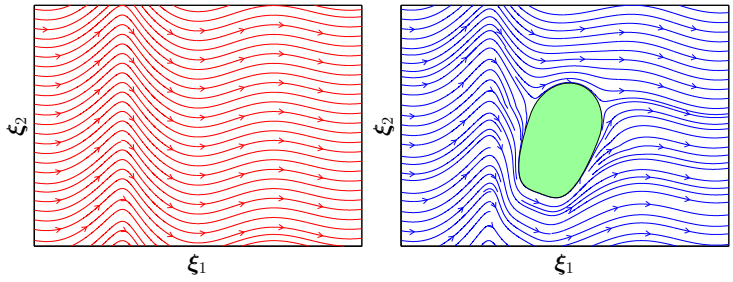
6.2 Robot Discrete Movements

So far we have shown how the dynamic modulation matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ can be used to deform a robot motion such that it does not collide with an obstacle. However in many robot experiments, e.g. reaching a target, not only should the robot avoid the obstacle, but it should also reach a target. In other words, we would like the modified motion to preserve the convergence property of the original dynamics while still ensuring that the motion does not penetrate the object. In this section we discuss the stability of DS when they are modulated with the proposed obstacle avoidance method. Throughout this section, we will assume that the target point $\boldsymbol{\xi}^*$ is outside the obstacle boundary, i.e. $\boldsymbol{\xi}^* \in \boldsymbol{\mathcal{X}}^f$.

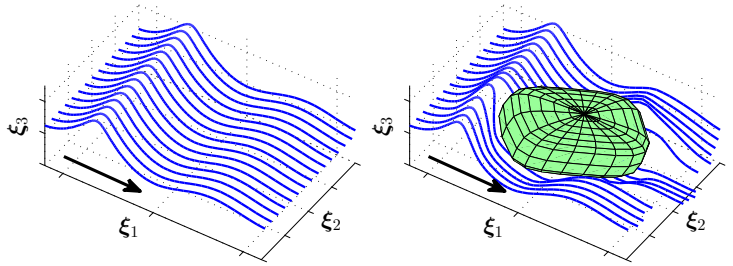
Suppose a d-dimensional globally asymptotically stable autonomous or non-autonomous DS defined by Eq. (4.31). The global stability of $\boldsymbol{f}$ requires that the velocity vanishes solely at the target point $\boldsymbol{\xi}^*$, i.e. $\boldsymbol{f}(\boldsymbol{\xi}^*) = \mathbf{0}$ for autonomous DS and $\lim_{t\to\infty} \boldsymbol{f}(t, \boldsymbol{\xi}^*) = \mathbf{0}$ for non-autonomous DS. When $\boldsymbol{f}(.)$ is modulated with the dynamic modulation matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$, $\boldsymbol{\xi}^*$ remains an equilibrium point because the velocity still vanishes at the target, i.e. $\boldsymbol{M}(\tilde{\boldsymbol{\xi}}^*)\boldsymbol{f}(\boldsymbol{\xi}^*) = \mathbf{0}$ for autonomous DS, and $\lim_{t\to\infty} \boldsymbol{M}(\tilde{\boldsymbol{\xi}}^*)\boldsymbol{f}(t, \boldsymbol{\xi}^*) = \boldsymbol{M}(\tilde{\boldsymbol{\xi}}^*) \lim_{t\to\infty} \boldsymbol{f}(t, \boldsymbol{\xi}^*) = \mathbf{0}$ for non-autonomous DS.

However, in the presence of an obstacle, the target may not remain the unique equilibrium point of the system. Other possible equilibrium points may be created due to the modulation term $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$. These points can be computed by looking at the null space of $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$. For all $\boldsymbol{\xi} \in \boldsymbol{\mathcal{X}}^f$, the matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ is full rank and hence $\boldsymbol{\xi}^*$ will be *the only equilibrium point* in $\boldsymbol{\mathcal{X}}^f$. Only on the boundaries of the obstacle, i.e. $\boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b$, $\boldsymbol{M}(\tilde{\boldsymbol{\xi}}^b)$ loses one rank yielding a number of spurious equilibrium points. In fact, these spurious equilibrium points $\boldsymbol{\xi}^s \in \boldsymbol{\mathcal{X}}^b$ are generated when there is collinearity between the velocity and the normal vector at the boundary points[4]:

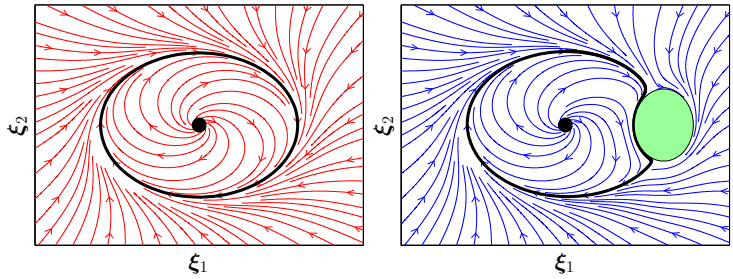
[4]From Theorem 6.2 we know that the normal velocity at the boundary points vanishes. Hence, if $\boldsymbol{f}(\boldsymbol{\xi})$ is aligned with the normal vector of the tangential hyperplane at a boundary point, we have $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})\boldsymbol{f}(\boldsymbol{\xi}) = \mathbf{0}$.



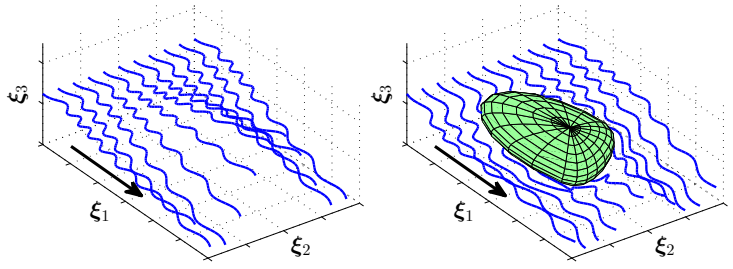
(a) A 2D autonomous flow defined by $\dot{\boldsymbol{\xi}}_1 = \log((\boldsymbol{\xi}_1 + 3)^2 + 2)$ and $\dot{\boldsymbol{\xi}}_2 = \sin(\boldsymbol{\xi}_1)$.



(b) A 3D autonomous flow defined by $\dot{\boldsymbol{\xi}}_1 = \log((\boldsymbol{\xi}_1 + 3)^2 + 2)$, $\dot{\boldsymbol{\xi}}_2 = 0$, and $\dot{\boldsymbol{\xi}}_3 = \sin(\boldsymbol{\xi}_1)$.



(c) A 2D stable limit-cycle defined by $\dot{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_2 - \boldsymbol{\xi}_1(\boldsymbol{\xi}_1^2 + \boldsymbol{\xi}_2^2 - 1)$ and $\dot{\boldsymbol{\xi}}_2 = -\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2(\boldsymbol{\xi}_1^2 + \boldsymbol{\xi}_2^2 - 1)$. The thick black line represents the stable limit cycle.



(d) A 3D non-autonomous flow defined by $\dot{\boldsymbol{\xi}}_1 = \log((\boldsymbol{\xi}_1 + 3)^2/(t+1) + 2)$, $\dot{\boldsymbol{\xi}}_2 = \sin(5t) - 0.1$, and $\dot{\boldsymbol{\xi}}_3 = 0.05t\cos(\boldsymbol{\xi}_2)$.

Figure 6.3: Modifying the original motion of a flow with a modulation matrix. In all four cases the obstacle is centered at $\boldsymbol{\xi}^o = \mathbf{0}$.

Algorithm 6.1 Procedure to handle equilibrium points at the obstacle boundary

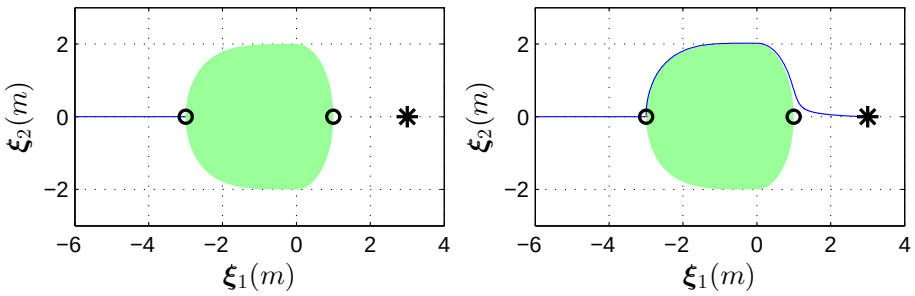
Input: $\boldsymbol{\xi}^t$, $\dot{\boldsymbol{\xi}}^t$, and the integration time step δt
if $\Gamma(\tilde{\boldsymbol{\xi}}^t) = 1$ and $\dot{\boldsymbol{\xi}}^t = 0$ **then**
  Choose one of the basis vectors $\boldsymbol{e}^i$ of the tangential hyper-plane.
  Define a small positive scalar $\alpha > 0$
  while true **do**
    $\boldsymbol{\xi}^{t+1} \leftarrow \boldsymbol{\xi}^t + \alpha \boldsymbol{R}\boldsymbol{e}^i \delta t$
    Compute $\dot{\boldsymbol{\xi}}^{t+1}$ from Eq. (6.17)
    if $(\boldsymbol{e}^i)^T \boldsymbol{R}^T \dot{\boldsymbol{\xi}}^{t+1} > 0$ **or** $\boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T \boldsymbol{R}^T \dot{\boldsymbol{\xi}}^{t+1} > 0$ **then**
      exit
    end if
    $t \leftarrow t + 1$
  end while
end if

$$\boldsymbol{n}(\tilde{\boldsymbol{\xi}}^s)^T \frac{\boldsymbol{R}^T \boldsymbol{f}(.)}{\|\boldsymbol{f}(.)\|} = \pm 1 \quad \text{and} \quad \Gamma(\tilde{\boldsymbol{\xi}}^s) = 1 \tag{6.18}$$

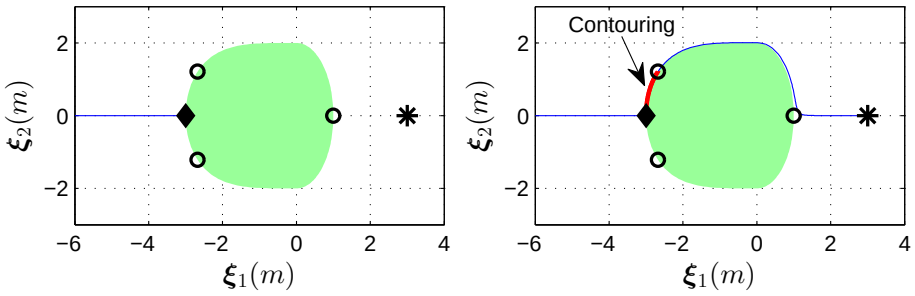
where $\boldsymbol{n}(\tilde{\boldsymbol{\xi}}^s)$ is the unit normal vector of the tangential hyperplane at $\tilde{\boldsymbol{\xi}}^s$. The set $\boldsymbol{\mathcal{X}}^s$ includes all solutions to Eq. (6.18). Depending on the function $\boldsymbol{f}(.)$, these equilibrium points could be either saddle points and/or local minima.

Computing this set of equilibrium points may not always be feasible. We can however simplify our task by considering that: "As all the equilibrium points appear solely on the obstacle boundary, one may avoid remaining stuck by using some external mechanisms". Algorithm 6.1 describes such a mechanism: when one detects that the motion has stopped at the outer surface (boundary) of an obstacle (i.e. at an equilibrium point), she applies a small perturbation along any of the basis vectors $\boldsymbol{e}^1..\boldsymbol{e}^{d-1}$. All of these vectors determine directions that ensure that the flow will move away from the obstacle. If the equilibrium point is a saddle point, the algorithm exits in one iteration. But if it is a local minimum, the obstacle is contoured along the direction of the basis vector $\boldsymbol{e}^i$ until it leaves the basin of attraction of the local minimum.

A positive scalar α can be used to control the amplitude of the movement along the basis vector $\boldsymbol{e}^i$. The value of α should be chosen by compromising between the accuracy, safety, and speed of the movement. For a large integration time step δt, one should use a small α to decrease the drifting error (due to integration) from the desired trajectory when contouring the obstacle. Furthermore, since contouring takes place at the outer surface of the obstacle, for safety reasons one should generally avoid selecting a high value for α. A very small value for α is also not recommended since it significantly slows down the contouring speed. Figure 6.4 illustrates two examples where Algorithm 6.1 is used to handle a saddle point and a local minimum.



(a) When the DS is defined by $\dot{\xi}_1 = -\xi_1 + 3$ and $\dot{\xi}_2 = -\xi_2$, the modulated dynamics has two saddle points at $(-3, 0)$ and $(0, 1)$. Without using Algorithm 6.1, the motion stops at $(-3, 0)$ (left graph). However, by using Algorithm 6.1 for one iteration, the motion continues until it reaches the target (right graph).



(b) By modifying the DS along its second dimension to $\dot{\xi}_2 = -3\xi_2$, the modulated dynamics will have one local minimum at $(-3, 0)$ and three saddle point at $(0, 1)$, $(-2.6757, 1.2120)$, and $(-2.6757, -1.2120)$. Without using Algorithm 6.1, the motion stops at the local minimum $(-3, 0)$ (left graph). In this situation, Algorithm 6.1 is used iteratively until the trajectory leaves the basin of attraction of the local minimum (i.e. the range between the local minimum and the saddle point). Then, the motion continues its way to the target (right graph). The part of trajectory that generated by Algorithm 6.1 is plotted with a thick red line.

Figure 6.4: Illustration of using Algorithm 6.1 to avoid possible equilibrium point(s) on the obstacle boundary. The target point is shown with a black star. The saddle point(s) and local minimum are represented with hollow circle and diamond, respectively. The obstacle boundary is modeled with $(\tilde{\xi}_1/1)^2 + (\tilde{\xi}_2/2)^2 = 1$ when $\tilde{\xi}_1 > 0$ and $(\tilde{\xi}_1/3)^4 + (\tilde{\xi}_2/2)^2 = 1$ elsewhere.

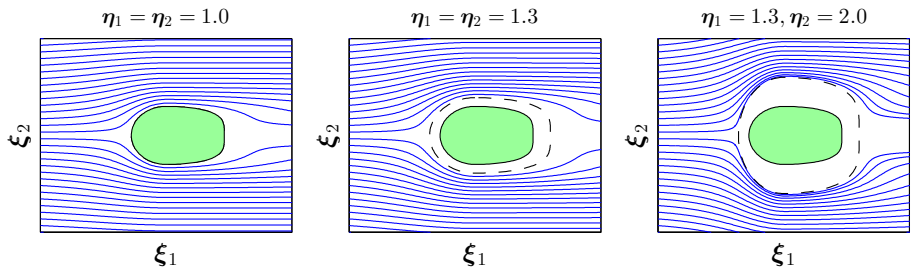


Figure 6.5: Controlling the safety margin around the obstacle via the safety factor. The obstacle is inflated in the direction $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ with the value $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$, respectively. The area between the dashed line and the obstacle boundary is the safety margin. The direction of the motion is from left to right.

6.3 Characterizing the Path during Obstacle Avoidance

When doing obstacle avoidance, sometimes it is more practical to customize the path to avoid an obstacle based on the object's property. For example, fragile or sharp objects may require a large safety margin while soft and round object may not. Furthermore, it is essential to react and deflect the robot trajectory earlier when it goes toward a fire flame than when it is just heading towards a soft pillow. In this section, we extend the proposed obstacle avoidance approach to incorporate user defined preferences during obstacle avoidance.

6.3.1 Safety Margin

The desired safety margin around an object can be obtained by scaling the state variable (in the obstacle frame of reference) in the dynamic modulation matrix $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ as follows:

$$\boldsymbol{M}(\tilde{\boldsymbol{\xi}_\eta}) = \boldsymbol{R}\boldsymbol{E}(\tilde{\boldsymbol{\xi}_\eta})\, \boldsymbol{D}(\tilde{\boldsymbol{\xi}_\eta})\, \boldsymbol{E}(\tilde{\boldsymbol{\xi}_\eta})^{(-1)} \boldsymbol{R}^T \tag{6.19}$$

where $\tilde{\boldsymbol{\xi}_{\boldsymbol{\eta}}} = \tilde{\boldsymbol{\xi}}./\boldsymbol{\eta}$ corresponds to the element-wise division of $\tilde{\boldsymbol{\xi}}$ by $\boldsymbol{\eta} \in \mathbb{R}^d$, and $\boldsymbol{\eta}_i \geq 1$, $\forall i \in 1..d$ is the desired safety factor, which inflates the object along each direction $\tilde{\boldsymbol{\xi}}_i$ with the magnitude $\boldsymbol{\eta}_i$ (in the obstacle frame of reference). By choosing different value for each $\boldsymbol{\eta}_i$, one can control the required safety margin along the corresponding direction of the object. Figure 6.5 illustrates the effect of different safety margins for a 2D object in a uniform flow[5].

[5]One can also define different safety factors along the positive and negative directions of each object's axis by considering an *if-else* condition on the sign of each $\tilde{\boldsymbol{\xi}}_i$.

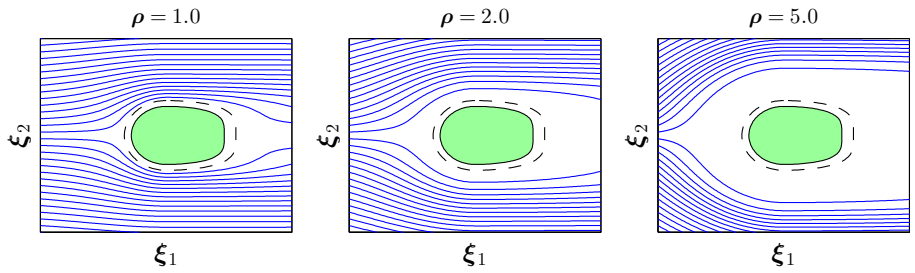


Figure 6.6: Controlling the reactivity of the motion to the presence of the obstacle (for $\eta_1 = \eta_2 = 1.2$). By increasing ρ, the reactivity increases, hence the flow deflects earlier in time and with a higher magnitude. Note that on the right graph, the white gap between the dashed line and the trajectories is part of the free region.

6.3.2 Reactivity

The magnitude of the modulation created by the obstacle can be tuned by modifying the eigenvalues of the dynamic modulation matrix as follows:

$$\begin{cases} \lambda^1(\tilde{\boldsymbol{\xi}}) = 1 - \frac{1}{|\Gamma(\tilde{\boldsymbol{\xi}})|^{\frac{1}{\rho}}} \\ \lambda^i(\tilde{\boldsymbol{\xi}}) = 1 + \frac{1}{|\Gamma(\tilde{\boldsymbol{\xi}})|^{\frac{1}{\rho}}} \qquad 2 \leq i \leq d \end{cases} \tag{6.20}$$

where $\rho > 0$ is the reactivity parameter. The larger the reactivity, the larger the amplitude of the deflection, and consequently the earlier the robot responds to the presence of an obstacle. A large ρ also extends the deflection farther out. Figure 6.6 illustrates the effect of using different reactivity parameters for a 2D object in a uniform flow.

6.3.3 Tail-Effect

In the proposed obstacle avoidance formulation, the modulation due to the obstacle continues affecting the motion even when the robot is moving away from the obstacle (see Fig. 6.7a). We call this effect of the obstacle on trajectories *tail-effect*. In some cases such a behavior may be beneficial as it approximately brings back the robot to the path it follows before facing the obstacle. When it is not desirable, one can remedy the tail-effect by defining the first eigenvalue of the dynamic modulation matrix as follows:

$$\lambda^1(\tilde{\boldsymbol{\xi}}) = \begin{cases} 1 - \frac{1}{|\Gamma(\tilde{\boldsymbol{\xi}})|^{\frac{1}{\rho}}} & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T \dot{\boldsymbol{\xi}} < 0 \\ 1 & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T \dot{\boldsymbol{\xi}} \geq 0 \end{cases} \tag{6.21}$$

In the above equation, we use the sign of $\boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T \dot{\boldsymbol{\xi}}$ to check whether a trajectory is going towards (negative sign) or away (positive sign) from the obstacle. Figure 6.7b illustrates the result after using Eq. (6.21). In this figure one can see that the tail-effect is significantly reduced. However, the slight modulation of the trajectories after passing the obstacle is still required in order to ensure the continuity in the velocity.

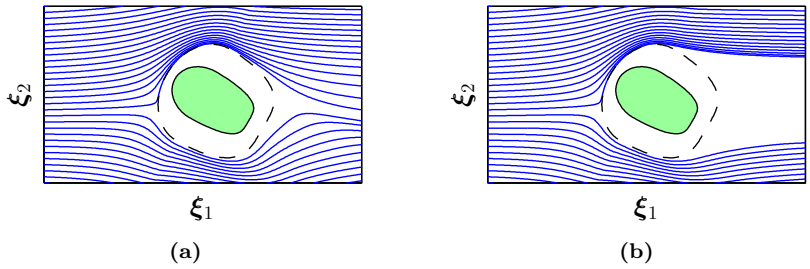


Figure 6.7: Controlling the tail-effect after passing the obstacle. **(a)** The tendency of the trajectories to follow the obstacle shape after passing it. **(b)** Remedying the tail-effect by defining the first eigenvalue according to Eq. (6.21).

6.4 Extension to Multiple Obstacles

So far we have shown how the dynamic modulation matrix can be used to avoid a single obstacle. However, in the presence of multiple obstacles, the current dynamic modulation matrix is ineffective and should be modified to include the effect of all the obstacles. Beware that this extension *cannot* be simply obtained by multiplying together the dynamic modulation matrix of all the obstacles. In this case, the impenetrability condition is only guaranteed for one of the obstacles. Note that for the sake of clarity of equations, in this section we did not consider the extensions that we have provided in Section 6.3 on the safety margin, reactivity, and tail-effect (here we use the default value $\eta = \rho = 1$, and do not remedy the tail-effect). In Section 6.6, we unify all these extensions into a single final model (see Table 6.1).

Let us consider K obstacles with associated reference points $\boldsymbol{\xi}^{o,k}$ and boundary functions $\Gamma^k(\boldsymbol{\xi}; \boldsymbol{\xi}^{o,k})$, $k = 1..K$ (the parameters of the k-th obstacle is denoted by $(.)^k$). We modify Eq. (6.16), and compute the eigenvalues of the k-th obstacle based on both its current state, and the state of other obstacles as follows:

$$\begin{cases} \lambda_1^k(\tilde{\boldsymbol{\xi}}^k) = 1 - \frac{\omega^k(\tilde{\boldsymbol{\xi}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}^k)|} \\ \lambda_i^k(\tilde{\boldsymbol{\xi}}^k) = 1 + \frac{\omega^k(\tilde{\boldsymbol{\xi}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}^k)|} \qquad 2 \leq i \leq d \end{cases} \tag{6.22}$$

where $\tilde{\boldsymbol{\xi}}^k = (\boldsymbol{R}^k)^T(\boldsymbol{\xi} - \boldsymbol{\xi}^{o,k})$, $\Gamma^k(\boldsymbol{\xi}^k)$ is the simplified notation of $\Gamma^k(\boldsymbol{\xi}; \boldsymbol{\xi}^{o,k})$, and $\omega^k(\tilde{\boldsymbol{\xi}}^k)$ are weighting coefficients that are computed according to:[6]

$$\omega^k(\tilde{\boldsymbol{\xi}}^k) = \prod_{i=1, i\neq k}^{K} \frac{(\Gamma^i(\tilde{\boldsymbol{\xi}}^i) - 1)}{(\Gamma^k(\tilde{\boldsymbol{\xi}}^k) - 1) + (\Gamma^i(\tilde{\boldsymbol{\xi}}^i) - 1)} \tag{6.23}$$

First observe that $\omega^k(\tilde{\boldsymbol{\xi}}^k)$ are continuous positive scalars between zero and one, i.e. $0 \leq \omega^k(\tilde{\boldsymbol{\xi}}^k) \leq 1$. Second, at the boundary of the k-th obstacle (i.e. $\Gamma^k(\tilde{\boldsymbol{\xi}}^k) = 1$), we have $\omega^k(\tilde{\boldsymbol{\xi}}^k) = 1$ and $\omega^i(\tilde{\boldsymbol{\xi}}^i) = 0$, $\forall i \in 1..K$ and $i \neq k$. As we

[6] Eq. (6.23) is in spirit very similar to the weighting coefficients proposed in (Waydo & Murray, 2003) with the difference that we use $\Gamma^k(\boldsymbol{\xi})$ to compute weights (rather than the distance between the obstacles).

will discuss later on, these two properties are crucial to ensure impenetrability of all the obstacles. Note that, when only one obstacle exists ($K = 1$), we simply set $\omega^1(\tilde{\boldsymbol{\xi}}^1) = 1$ and Eq. (6.22) simplified into Eq. (6.16).

By substituting Eq. (6.23) into the matrix of eigenvalues given by Eq. (6.15), the dynamic modulation matrix for each obstacle becomes:

$$\boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}^k) = \boldsymbol{R}^k \boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}^k)\, \boldsymbol{D}^k(\tilde{\boldsymbol{\xi}}^k)\, \left(\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}^k)\right)^{-1} \left(\boldsymbol{R}^k\right)^T \tag{6.24}$$

The combined modulation matrix that considers the net effect of all the obstacles is then given by:

$$\bar{\boldsymbol{M}}(\boldsymbol{\xi}) = \prod_{k=1}^{K} \boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}^k) \tag{6.25}$$

Eq. (6.25) ensures the impenetrability of all the K obstacles. To verify this, suppose a point $\boldsymbol{\xi}^b$ on the boundary of the k-th obstacle. At this point, following the properties of ω mentioned above and considering Eqs. (6.15), (6.22), (6.24) and (6.25), we have:

$$\begin{aligned}
\omega^i(\tilde{\boldsymbol{\xi}}^{b,i}) = 0 \quad &\Rightarrow \quad \lambda^i_j(\tilde{\boldsymbol{\xi}}^{b,i}) = 1 \qquad \forall j \in 1..d, \forall i \in 1..K, i \neq k \\
&\Rightarrow \quad \boldsymbol{D}^i(\tilde{\boldsymbol{\xi}}^{b,i}) = \boldsymbol{I} \\
&\Rightarrow \quad \boldsymbol{M}^i(\tilde{\boldsymbol{\xi}}^{b,i}) = \boldsymbol{R}^i \boldsymbol{E}^i(\tilde{\boldsymbol{\xi}}^{b,i})\, \boldsymbol{I}\, \left(\boldsymbol{E}^i(\tilde{\boldsymbol{\xi}}^{b,i})\right)^{-1} \left(\boldsymbol{R}^i\right)^T \\
&\Rightarrow \quad \boldsymbol{M}^i(\tilde{\boldsymbol{\xi}}^{b,i}) = \boldsymbol{R}^i\, \boldsymbol{I}\, \left(\boldsymbol{R}^i\right)^T = \boldsymbol{I} \\
&\Longrightarrow \quad \bar{\boldsymbol{M}}(\boldsymbol{\xi}^b) = \boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}^{b,k})
\end{aligned}$$

Furthermore, because $\omega^k(\tilde{\boldsymbol{\xi}}^{b,k}) = 1$, $\boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}^{b,k})$ and by extension $\bar{\boldsymbol{M}}(\boldsymbol{\xi}^b)$ is exactly similar to Eq. (6.13). Hence following Theorem 6.2, the obstacle is impenetrable. By moving from one obstacle to another, the weighting coefficients smoothly changes between zero and one, and by this, the impenetrability is always ensured for all the obstacles.

Following the discussion given in Section 6.2, the target point $\boldsymbol{\xi}^*$ is the only equilibrium point in the free region because each of the modulation matrix $\boldsymbol{M}^k$ has full rank. However, as discussed before, on the boundaries of each obstacle a set of saddle points or local minima may be generated. Provided the obstacles are not connected, i.e. they do not have a contact point, these equilibrium points can be handled by following Algorithm 6.1.

Fig. 6.8 illustrates the implementation of Eq. (6.25) in the presence of five obstacles positioned in different ways. To simplify the reference to these objects, they are numbered from one to five. In this figure, the thick black line is the streamline that starts on the symmetric line of the obstacles arrangement. As can be seen, the combined modulation matrix is able to prevent hitting the obstacles even if there is a narrow passage between them (see for example Figs. 6.8a to 6.8c).

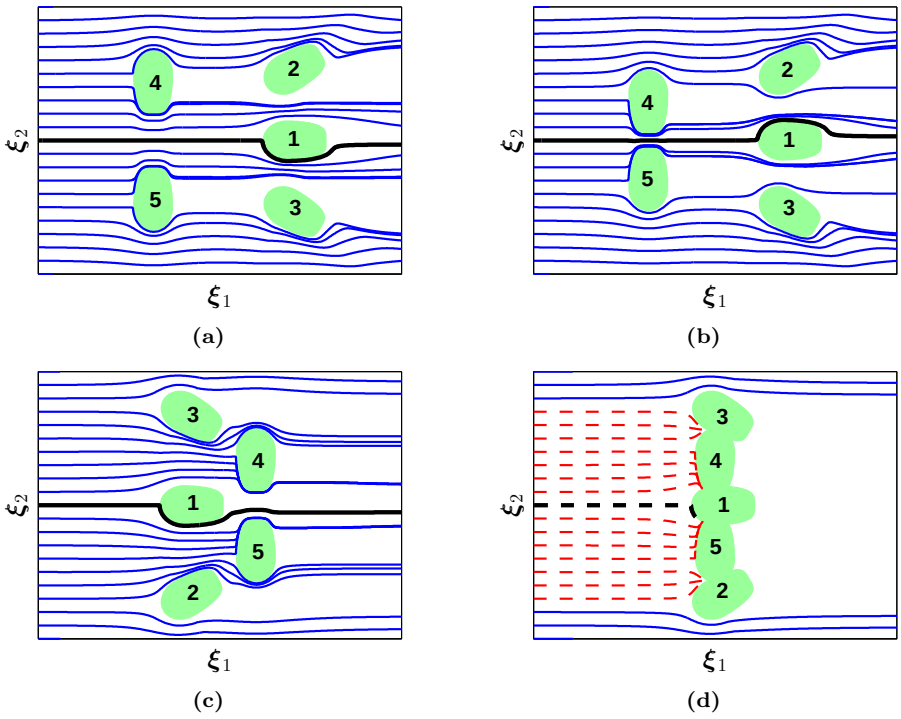


Figure 6.8: Extension of the proposed approach to multiple obstacles. The combined dynamic modulation matrix ensures the impenetrability of all obstacle even if they are very close or connected to each other. However, for the case where the objects are connected (see **(d)**), some local minima may appear that cannot be avoided with Algorithm 6.1. Trajectories that stop at the local minima are plotted with dashed lines. A trivial solution to handle this problem is to model all the connected obstacles as a single convex obstacle.

Fig. 6.8d shows the result for the case where all obstacles are connected. First observe that the resulting shape is no longer convex, but the impenetrability of the obstacles is still preserved. However in the presence of the resulting concave shape, Algorithm 6.1 cannot be used to avoid local minima. A trivial solution to handle this problem is to model all the connected obstacles as a single convex obstacle. Note that at the boundaries' intersection points, the weighting coefficients ω^k are undefined (because the distance to more than one obstacle is zero, and thus a division by zero occurs). At these points, we have simply stopped the simulation.

6.5 Extension to Moving Obstacles

So far we have considered situations where obstacles are static. In this section we extend our formulation to perform obstacle avoidance in the presence of multiple moving obstacles with linear and/or rotational velocities. In the presence of a single obstacle, this extension is straight forward and can be achieved by computing the modulation in the obstacle's frame of reference. Suppose an obstacle $\Gamma(\tilde{\boldsymbol{\xi}})$ that is moving with linear and rotational velocities $\dot{\boldsymbol{\xi}}_L^o$ and $\dot{\boldsymbol{\xi}}_R^o$, respectively. The modulated dynamics for obstacle avoidance becomes:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{M}(\tilde{\boldsymbol{\xi}})\big(\boldsymbol{f}(.) - \dot{\boldsymbol{\xi}}^o_L - \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}}\big) + \dot{\boldsymbol{\xi}}^o_L + \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}} \tag{6.26}$$

where $(.) \times (.)$ denotes the cross product and $\boldsymbol{M}(\tilde{\boldsymbol{\xi}})$ is the modulation given by Eq. (6.13). In this equation, the term $\boldsymbol{f}(.) - \dot{\boldsymbol{\xi}}^o_L - \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}}$ transforms the velocity of the robot to the obstacle's coordinates system. Then, the modulation is performed in this coordinates system where the object is static, yet the robot is moving with a different speed. After applying the modulation, the result is transformed back to the world's frame of reference through the last term.

Equation (6.26) ensures impenetrability of a single moving obstacle. To verify this, suppose a point $\boldsymbol{\xi}^b$ on the boundary of the moving obstacle at time t. As outlined before, the modulation cancels the radial velocity of the robot at the boundary (hence the robot can only move along the tangential hyperplane). But this is not enough as the robot may hit the obstacle in the next moment t^+ since the obstacle is moving. However, this can be avoided by adding the instant velocity of the point $\boldsymbol{\xi}^b$ due to obstacle motion, which is given by $\dot{\boldsymbol{\xi}}^o_L + \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}}^b$, to the modulated velocity.

As a side effect, Eq. (6.26) could induce some unnecessary movements to the robot even when the robot is far from the obstacle (note that the angular velocity grows proportionally with $\|\tilde{\boldsymbol{\xi}}\|$). This can be tackled by adding an exponential term that diminishes the induced velocity due to the obstacle's movement as $\|\tilde{\boldsymbol{\xi}}\|$ increases:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{M}(\tilde{\boldsymbol{\xi}})\Big(\boldsymbol{f}(.) - e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)}\big(\dot{\boldsymbol{\xi}}^o_L + \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}}\big)\Big) + e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)}\big(\dot{\boldsymbol{\xi}}^o_L + \dot{\boldsymbol{\xi}}^o_R \times \tilde{\boldsymbol{\xi}}\big) \tag{6.27}$$

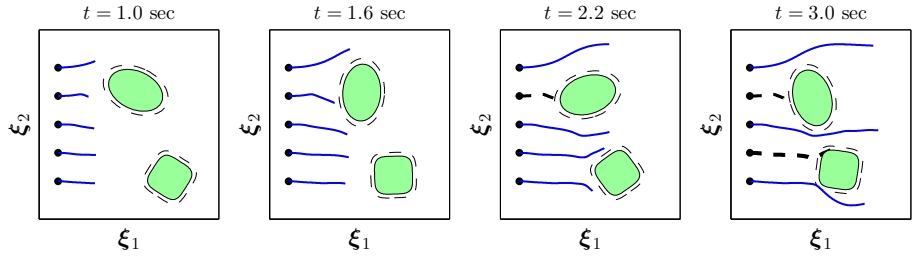
where σ^o is a positive scalar controlling the rate of decay of the exponential term. The higher the σ^o, the earlier the robot responds to the obstacle motion. The above change does not compromise impenetrability of the obstacle as on the boundary of the obstacle we have $e^{-\frac{1}{\sigma^o}(\Gamma(\tilde{\boldsymbol{\xi}})-1)} = 1$.

In the presence of multiple moving obstacles, further considerations should be taken so that the above transformation smoothly shift from one obstacle to another based on the current position of the robot. To achieve this goal, we follow the same principle as the one described in Section 6.4 by using some weighting coefficients to control the priorities of obstacles.

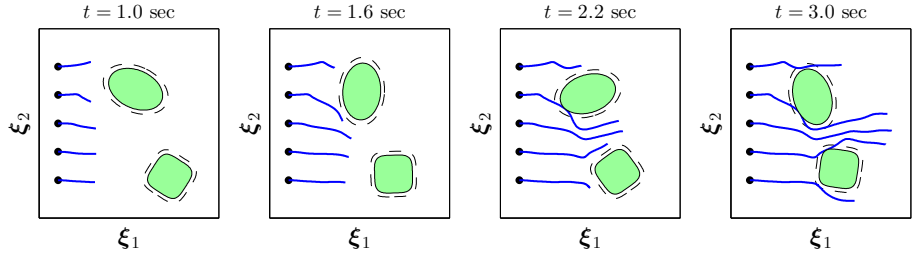
Let us consider K disconnected obstacles that are described by $\Gamma^k(\tilde{\boldsymbol{\xi}}^k)$, $k \in 1..K$, with the associated translational and rotational velocities $\dot{\boldsymbol{\xi}}^{o,k}_L$ and $\dot{\boldsymbol{\xi}}^{o,k}_R$, respectively. We define the net shift in velocity $\bar{\dot{\boldsymbol{\xi}}}^o$ due to the presence of these obstacles with:

$$\bar{\dot{\boldsymbol{\xi}}}^o = \sum_{k=1}^{K} \dot{\boldsymbol{\xi}}^{o,k} = \sum_{k=1}^{K} e^{-\frac{1}{\sigma^{o,k}}(\Gamma^k(\tilde{\boldsymbol{\xi}}^k)-1)}\omega^k(\tilde{\boldsymbol{\xi}}^k)\big(\dot{\boldsymbol{\xi}}^{o,k}_L + \dot{\boldsymbol{\xi}}^{o,k}_R \times \tilde{\boldsymbol{\xi}}^k\big) \tag{6.28}$$

where $\omega^k(\tilde{\boldsymbol{\xi}}^k)$ are computed according to Eq. (6.23). In case the tail effect is not desired (i.e. $\kappa = 0$), one could remove the modulation effect by setting $\dot{\boldsymbol{\xi}}^{o,k} = 0$ for each obstacle that is moving away from the robot (i.e. when $\big(\dot{\boldsymbol{\xi}}^{o,k}\big)^T \tilde{\boldsymbol{\xi}}^{o,k} < 0$).



(a) Without considering the obstacles' motion (the quasi-static case). The dashed black lines show the failure cases where the robot actually collides with the obstacles.



(b) With considering the obstacles' motion. In this case, collision avoidance for all trajectories is ensured.

Figure 6.9: Illustration of the obstacle avoidance in the presence of two moving obstacles. As we can see, solely in the dynamic case, where the obstacles' motion is considered, the trajectories can safely pass the obstacles. In this example, the trajectories move from left to right with $\dot{\boldsymbol{\xi}} = [2; 0]$ m/s. The oval-shaped object has the linear velocity $\dot{\boldsymbol{\xi}}_L^{o,1} = [-0.4; -0.2]$ m/s and the rotational velocity $\dot{\boldsymbol{\xi}}_R^{o,1} = -2$ rad/s. These values for the square-shaped obstacle are $[-0.4; -0.2]$ m/s and 1 rad/sec. Both objects have the safety factor of $\boldsymbol{\eta} = 1.2$. The variance σ^o is set to 2 and 10 for the oval and square-shaped obstacles, respectively.

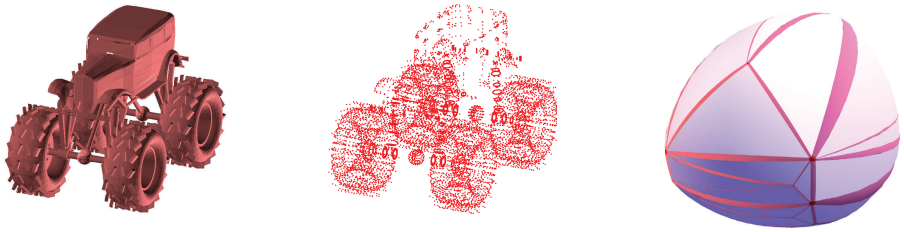
The combined modulation that considers the net effect of all moving/static obstacles is then given by:

$$\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\boldsymbol{\xi})\big(\boldsymbol{f}(.) - \bar{\dot{\boldsymbol{\xi}}}^o\big) + \bar{\dot{\boldsymbol{\xi}}}^o \tag{6.29}$$

where $\bar{\boldsymbol{M}}(\boldsymbol{\xi})$ is given by Eq. (6.25). Equation (6.29) ensures the impenetrability of all the K obstacles. For a point $\boldsymbol{\xi}^b$ on the boundary of the k-th obstacle, only $\omega^k = 1$ and all the other weighting coefficients are zero. Hence $\bar{\boldsymbol{M}}(\boldsymbol{\xi}^b) = \boldsymbol{M}^k(\boldsymbol{\xi}^b)$ and $\bar{\dot{\boldsymbol{\xi}}}^o = \dot{\boldsymbol{\xi}}_L^{o,k} + \dot{\boldsymbol{\xi}}_R^{o,k} \times \tilde{\boldsymbol{\xi}}^{b,k}$, and thus the obstacle is impenetrable. Similarly to the static case, by moving from one obstacle to another, the weighting coefficients smoothly change between zero and one, and by this, impenetrability is always ensured for all the obstacles.

Figure 6.9 shows an example of obstacle avoidance in the presence of two moving obstacles. It compares two situations: 1) The quasi-static case where the obstacles' motion are neglected, and the modulation is computed at each time based on the instantaneous position and orientation of the obstacles (see Fig. 6.9a), and 2) The dynamic case where the obstacles' motion are taken into account (see Fig. 6.9b). As we can see, in the quasi-static case the impenetrability of the obstacles are no longer ensured, whereas in the dynamic case all the trajectories can safely pass the obstacles.

Figure 6.10: Illustration of two complex objects that are modeled with two smooth hypersurfaces. The analytical model for the drawer is $\Gamma(\tilde{\boldsymbol{\xi}})$: $(\tilde{\boldsymbol{\xi}}_1/0.4)^4 + (\tilde{\boldsymbol{\xi}}_2/0.4)^8 + (\tilde{\boldsymbol{\xi}}_3/0.6)^4 = 1$, and the mug is modeled with $(\tilde{\boldsymbol{\xi}}_1/0.05)^4 + (\tilde{\boldsymbol{\xi}}_2/0.05)^8 + (\tilde{\boldsymbol{\xi}}_3/0.05)^4 = 1$ when $\tilde{\boldsymbol{\xi}}_2 > 0$ and $(\tilde{\boldsymbol{\xi}}_1/0.05)^4 + (\tilde{\boldsymbol{\xi}}_2/0.08)^2 + (\tilde{\boldsymbol{\xi}}_3/0.05)^4 = 1$ elsewhere.



(a) **(b)** **(c)**

Figure 6.11: Illustration of generating a BV from the point cloud of a toy car. **(a)** The 3D model of the car. **(b)** The point cloud of the car taken from the Princeton Shape Benchmark (Shilane et al., 2004). **(c)** The $\mathcal{C}^1$ smoothness BV generated using the method described by (Benallegue et al., 2009).

6.6 Obstacle Avoidance Module

The proposed obstacle avoidance algorithm requires a user to provide an analytical formulation of the outer surface of the obstacle. When provided with the 3D model of the object, one may compute a smooth convex envelope (also known as convex bounding volume) that fits tightly around the object. This Bounding Volume (BV) can be used (instead of the object's shape) to perform obstacle avoidance. Figure 6.10 illustrates such 3D convex envelopes generated from the 3D models of a mug and a drawer.

When solely the point cloud description of the object is available, one may use one of the estimation techniques to approximate the BV. For example, in (Benallegue et al., 2009), the BV is approximated using a set of spheres and tori. To use this method, one first needs to find the relevant patch (either sphere or torus) of the BV that corresponds to the current position of the robot. Then, based on the analytical formulation of that patch, one can compute the dynamic modulation matrix as described before. Recall that our obstacle avoidance module only requires the convexity and $\mathcal{C}^1$ smoothness of the BV, which are fulfilled in this work. Figure 6.11 shows an example of the convex BV generated from the point cloud of a toy car using the method above[7].

[7]The author would like to thank M. Benallegue and A. Kheddar for providing the source code of the STP-BV method to generate the BV from the point cloud of the object.

Table 6.1: Nomenclature of the presented DS-based obstacle avoidance method

Symbol	Description
d	Dimension of the state variable
K	Number of obstacles
$\boldsymbol{\xi} \in \mathbb{R}^d$	Current robot position
$\dot{\boldsymbol{\xi}} \in \mathbb{R}^d$	Current robot velocity
$\boldsymbol{\xi}^{o,k} \in \mathbb{R}^d$	Center of the k-th obstacle
$\tilde{\boldsymbol{\xi}}^k \in \mathbb{R}^d$	Relative position to the k-th obstacle
$\tilde{\boldsymbol{\xi}}^k_{\boldsymbol{\eta}} \in \mathbb{R}^d$	Scaled relative position to the k-th obstacle
$\dot{\boldsymbol{\xi}}^{o,k}_L \in \mathbb{R}^d$	Linear velocity of the k-th obstacle
$\dot{\boldsymbol{\xi}}^{o,k}_R \in \mathbb{R}^d$	Angular velocity of the k-th obstacle
$\Gamma^k : \mathbb{R}^d \mapsto \mathbb{R}$	Analytical description of the k-th obstacle
$\boldsymbol{E}^k \in \mathbb{R}^{d\times d}$	Matrix of Basis vectors of the k-th obstacle
$\boldsymbol{D}^k \in \mathbb{R}^{d\times d}$	Matrix of eigenvalues of the k-th obstacle
$\boldsymbol{M}^k \in \mathbb{R}^{d\times d}$	Dynamic Modulation matrix of the k-th obstacle
$\boldsymbol{R}^k \in \mathbb{R}^{d\times d}$	Rotation matrix of the k-th obstacle
$\boldsymbol{n}^k \in \mathbb{R}^d$	Normal vector of the deflection hyperplane for the k-th obstacle
$\boldsymbol{e}^{i,k} \in \mathbb{R}^d$	The i-th basis vector of the k-th obstacle
$\lambda^k_i \in [0\ \ 2]$	The i-th eigenvalue of the k-th obstacle
$\omega^k \in [0\ \ 1]$	Weighting coefficient of the k-th obstacle
$\boldsymbol{\eta} \in \mathbb{R}^d, \boldsymbol{\eta_i} \geq 1$	Safety factor
$\rho \in \mathbb{R}^+$	Reactivity
$\sigma^{o,k} \in \mathbb{R}^+$	Controlling responsiveness to the k-th obstacle movement
$\bar{\boldsymbol{M}}(\boldsymbol{\xi}) \in \mathbb{R}^{d\times d}$	Combined dynamic modulation matrix
$\bar{\dot{\boldsymbol{\xi}}}^o \in \mathbb{R}^d$	Net shift in velocity due to presence of moving obstacles

When doing obstacle avoidance in a dynamic environment, it is hardly possible to generate the BVs from the output of the vision system in realtime. Thus, it is necessary to generate a library that stores the analytical formulations of different objects. In our implementation, we rely on a library of objects with known analytical convex envelopes. We use this analytical descriptor of the envelop both to detect the object and for our obstacle avoidance module.

A summary of the presented obstacle avoidance method is provided in Algorithm 6.2. For clarity of the method, a complete list of the required variables and their description is summarized in Table 6.1. A conceptual sketch describing how the presented method can be used in robot experiments is also illustrated in Fig. 6.12. In this approach, first the raw output of the vision system is sent to an object recognition module to identify the object(s). When the objects are recognized, their corresponding properties such as the analytical formulation of the boundary, safety factor, etc. are sent to the obstacle avoidance module. The obstacle avoidance module modifies the original dynamics of the motion based on the combined dynamic modulation matrix $\bar{\boldsymbol{M}}(\boldsymbol{\xi})$ and the net shift in velocity $\bar{\dot{\boldsymbol{\xi}}}^o$ so as to avoid the obstacle safely.

In the presence of fast unknown moving obstacles, the object recognition phase may not provide the agility required to avoid the obstacle (especially when there is a large library of the objects). In these situations, it might be more adequate to replace the object recognition phase with an automatic BV

Algorithm 6.2 DS-Based Obstacle Avoidance. See Table 6.1 for the description of symbols.

Input: $\boldsymbol{\xi}$, $\boldsymbol{f}(.)$, and $\{\boldsymbol{R}^k, \boldsymbol{\eta}^k, \rho^k, \kappa^k, \sigma^{o,k}, \dot{\boldsymbol{\xi}}_L^{o,k}, \dot{\boldsymbol{\xi}}_R^{o,k}\}_{k=1}^K$

1: **for** each obstacle $k, k \in 1..K$ **do**

2: $\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k = \big((\boldsymbol{R}^k)^T(\boldsymbol{\xi} - \boldsymbol{\xi}^{o,k})\big)./\boldsymbol{\eta}^k$

3: $\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \Big[\boldsymbol{n}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \quad \boldsymbol{e}^{1,k}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \quad \cdots \quad \boldsymbol{e}^{d-1,k}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\Big]$

4: $\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{cases} 1 & \text{if } K = 1 \\ \prod_{i=1, i\neq k}^K \frac{(\Gamma^i(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)}{(\Gamma^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)+(\Gamma^i(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)-1)} & \text{otherwise} \end{cases}$

5: $\begin{cases} \lambda_1^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{cases} 1 - \frac{\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)|^{\frac{1}{\rho}}} & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T\dot{\boldsymbol{\xi}} < 0 \ \textbf{or} \ \kappa = 1 \\ 1 & \boldsymbol{n}(\tilde{\boldsymbol{\xi}})^T\dot{\boldsymbol{\xi}} \geq 0 \ \textbf{and} \ \kappa = 0 \end{cases} \\ \lambda_i^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = 1 + \frac{\omega^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)}{|\Gamma(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)|^{\frac{1}{\rho}}} \qquad 2 \leq i \leq d \end{cases}$

6: $\boldsymbol{D}(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \begin{bmatrix} \lambda_1^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \lambda_d^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) \end{bmatrix}$

7: $\boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k) = \boldsymbol{R}^k\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\,\boldsymbol{D}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\,\big(\boldsymbol{E}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)\big)^{-1}\big(\boldsymbol{R}^k\big)^T$

8: **end for**

9: $\bar{\boldsymbol{M}}(\boldsymbol{\xi}) = \prod_{k=1}^K \boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}_{\boldsymbol{\eta}}^k)$

10: $\bar{\dot{\boldsymbol{\xi}}}^o = \sum_{k=1}^K e^{-\frac{1}{\sigma^{o,k}}(\Gamma^k(\tilde{\boldsymbol{\xi}}^k)-1)}\omega^k(\tilde{\boldsymbol{\xi}}^k)\big(\dot{\boldsymbol{\xi}}_L^{o,k} + \dot{\boldsymbol{\xi}}_R^{o,k} \times \tilde{\boldsymbol{\xi}}^k\big)$

Output: $\dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\boldsymbol{\xi})\big(\boldsymbol{f}(.) - \bar{\dot{\boldsymbol{\xi}}}^o\big) + \bar{\dot{\boldsymbol{\xi}}}^o$

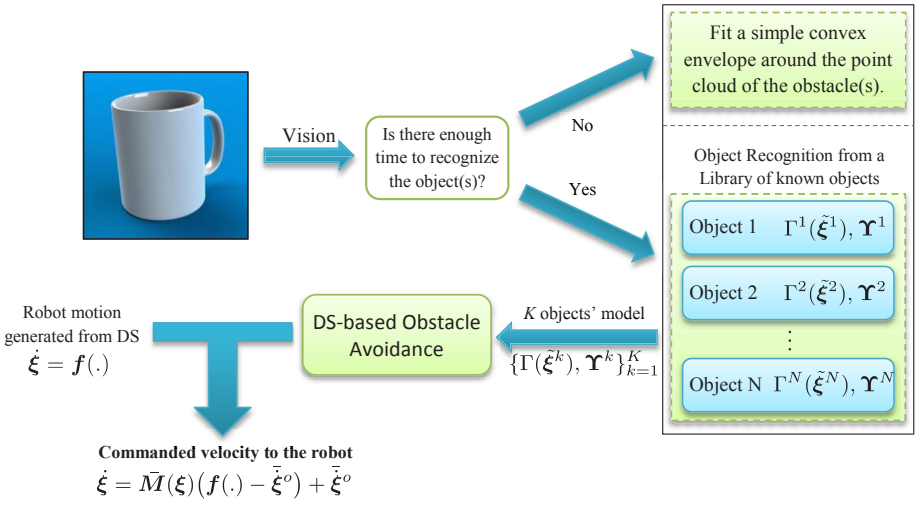


Figure 6.12: A conceptual sketch describing the implementation of the obstacle avoidance module for robot motions. The set $\boldsymbol{\Upsilon}^k = \{\boldsymbol{R}^k, \boldsymbol{\eta}^k, \rho^k, \kappa^k, \sigma^{o,k}, \boldsymbol{\xi}_L^{o,k}, \dot{\boldsymbol{\xi}}_R^{o,k}\}$ contains the corresponding properties of the k-th obstacle.

generator algorithm (see Fig. 6.12). Generating a simple BV (e.g. an ellipsoid) around the point cloud of an obstacle can be done quite quickly. If the object moves very rapidly, it is recommended to set a large value for the safety margin $\boldsymbol{\eta}$ and for the reactivity parameter ρ (see Section 6.3) to increase the robustness to uncertainties.

Furthermore, when there are many obstacles in the working space of the robot, it may not be necessary (and also computationally feasible) to track all the obstacles all the time. Since the modulation decreases as the distance to the obstacle increases, one could ignore all obstacles for which the associated modulation matrices are close to identity[8] (since we have $\lim_{\tilde{\boldsymbol{\xi}}^k \to \infty} \boldsymbol{M}^k(\tilde{\boldsymbol{\xi}}^k) = \boldsymbol{I}$).

By taking into account the obstacles that are locally relevant, the processing time for the vision systems could decrease significantly. However, this will be at the cost of imposing a small discontinuity in the robot velocity when an obstacle is added or removed from the set of relevant obstacles. By setting a small threshold, this discontinuity practically becomes very negligible.

6.7 Experiments

We evaluate the performance of the proposed approach in three ways: 1) On a set of theoretical autonomous and non-autonomous DS, 2) On a set of 2D motions described by dynamical systems that are inferred from human demonstrations, using two different learning approaches: SEDS and DMP (Ijspeert et al., 2002a), and 3) In robot experiments performed on the 7-DoF Barrett WAM and KUKA DLR arms. Unless otherwise specified, throughout this section we consider $\rho = \kappa = 1$, and the state of the system is defined as either planar or 3D motions, i.e. $\boldsymbol{\xi} = [x \;\; y]^T$ or $\boldsymbol{\xi} = [x \;\; y \;\; z]^T$ respectively.

6.7.1 Simulation Experiments on Theoretical DS

We first evaluate our method in simulation using the motion flow $\boldsymbol{f}(.)$ that is described by five different theoretical dynamical systems. These DS are defined in Table 6.2 and their phase plots are illustrated in Fig. 6.13.

The first DS is globally asymptotically stable at the origin. Due to the cosine term, this DS displays a high nonlinear behavior. The second DS is interesting in that it has infinite number of attractors, saddle points, and unstable points. The third DS has a stable limit cycle that includes an unstable point located at the origin. The forth DS is globally unstable and has a unique unstable point at the origin. Due to the sine terms, this DS also displays a high nonlinear behavior. The fifth DS is globally unstable without any equilibrium point.

[8] For example, we consider the k-th obstacle is locally relevant in the current position of the robot if: $|\lambda_i^k(\tilde{\boldsymbol{\xi}}^k) - 1| > \varsigma, \forall i = 1..d$, where ς is a small positive threshold.

Table 6.2: The theoretical DS used for the Simulation Experiments

(a) $\begin{cases} \dot{x} = -x \\ \dot{y} = -x\cos x - y \end{cases}$	**(d)** $\begin{cases} \dot{x} = y - x(x^2 + y\sin x - 1) \\ \dot{y} = -x - y(x^2 + y\sin x - 1) \end{cases}$
(b) $\begin{cases} \dot{x} = \cos x \\ \dot{y} = \sin y \end{cases}$	**(e)** $\begin{cases} \dot{x} = \lvert x\rvert/2 + 1 \\ \dot{y} = 0 \\ \dot{z} = \lvert y\rvert\cos t \end{cases}$
(c) $\begin{cases} \dot{x} = y \\ \dot{y} = -x + 0.9y(1 - x^2) \end{cases}$	

All these DS are evaluated in the presence of multiple obstacles. For simplicity, we consider two types of the 2D obstacles and one 3D obstacle, but we use them in different scales, orientations, and reference points. These obstacles are formulated as follows:

$$\text{Obstacle } \#1: \quad \Gamma(\tilde{\boldsymbol{\xi}}) = (\tilde{x}/0.75)^4 + (\tilde{y}/1)^2 = 1$$

$$\text{Obstacle } \#2: \quad \Gamma(\tilde{\boldsymbol{\xi}}) = \begin{cases} (\tilde{x}/1.2)^4 + (\tilde{y}/0.4)^2 = 1 & y \leq y^o \\ (\tilde{x}/1.2)^2 + (\tilde{y}/1)^2 = 1 & y > y^o \end{cases}$$

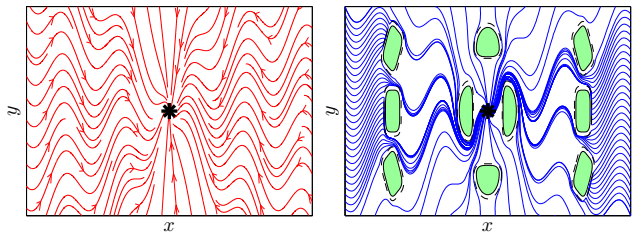
$$\text{Obstacle } \#3: \quad \Gamma(\tilde{\boldsymbol{\xi}}) = \begin{cases} \tilde{x}^2 + (\tilde{y}/1.4)^2 + (2\tilde{z})^2 = 1 & y \leq y^o \\ \tilde{x}^2 + \tilde{y}^4 + (2\tilde{z})^2 = 1 & y > y^o \end{cases}$$

Considering Fig. 6.13, all obstacles can be successfully avoided in all types of DS even in the presence of high nonlinearities and/or having several equilibrium points. As it is expected, the multiplication of the combined dynamic modulation matrix does not modify the original equilibrium points of the system, and does not add any extra equilibrium point in the free space $\bar{\boldsymbol{\mathcal{X}}}^f$. The potential spurious equilibrium points on the boundaries of obstacles are also handled using Algorithm 6.1.

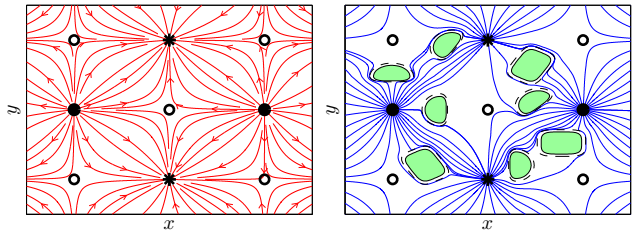
6.7.2 Simulation Experiments on SEDS and DMP

In this section we evaluate the performance of the proposed approach to generate handwritten trajectories forming the alphabet letters 'N', 'G' and 'J'. Each motion was demonstrated three times. They were collected at 50Hz from pen input using a Tablet-PC. The motions are learned using SEDS and DMP. As outlined before, SEDS builds an estimate of the motion through an autonomous DS $\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi})$, and thus in the presence of obstacle(s) it can be modulated by following Eq. (6.25), whereas DMP models a motion as a second order DS that takes the form of $\ddot{\boldsymbol{\xi}} = \boldsymbol{g}(t, \boldsymbol{\xi}, \dot{\boldsymbol{\xi}})$. This function can be transformed into a first order DS via:

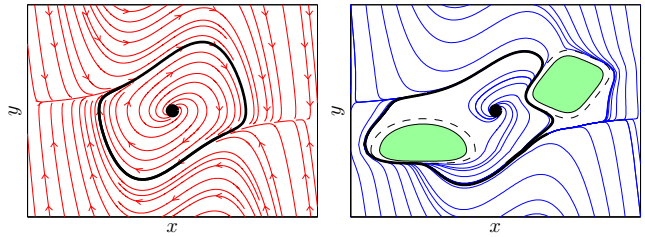
$$\begin{cases} \dot{\boldsymbol{\xi}} = \boldsymbol{\zeta} \\ \dot{\boldsymbol{\zeta}} = \boldsymbol{g}(t, \boldsymbol{\xi}, \boldsymbol{\zeta}) \end{cases} \tag{6.30}$$



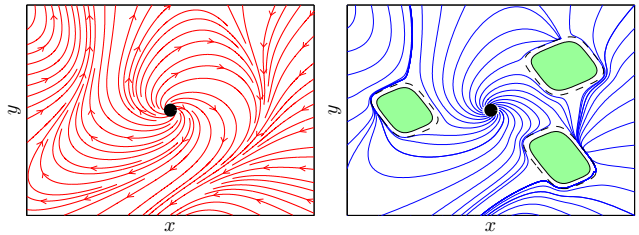
(a) Globally stable DS



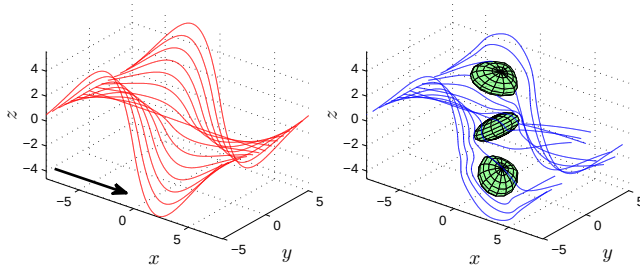
(b) Locally stable DS



(c) Stable limit cycle. In this graph, the thick black line corresponds to the stable limit cycle.



(d) Unstable DS



(e) Unstable DS

Figure 6.13: Performance evaluation of the proposed obstacle avoidance module in the presence of five complex DS. The left column shows the original DS, and the right column illustrates the modulated DS in the presence of multiple obstacles. In this figure, stable, unstable, and saddle points are shown in star, solid circle and hollow circle, respectively. Obstacles are colored in green and the black dashed lines illustrate their safety margin ($\boldsymbol{\eta} = 1.2$ is considered for all the obstacles). For formulation of the DS and the obstacles please refer to the text in Section 6.7.1.

and the modulation due to the presence of obstacle(s) can be obtained as follows[9]:

$$\begin{cases} \dot{\boldsymbol{\xi}} = \bar{\boldsymbol{M}}(\tilde{\boldsymbol{\xi}})\boldsymbol{\zeta} \\ \dot{\boldsymbol{\zeta}} = \boldsymbol{g}(t, \boldsymbol{\xi}, \bar{\boldsymbol{M}}(\tilde{\boldsymbol{\xi}})\boldsymbol{\zeta}) \end{cases} \tag{6.31}$$

Fig. 6.14 illustrates the results for these motions in the presence of four different obstacles. In this experiment the obstacles are modeled with the following formulations:

$$\textbf{Case 1: } \Gamma(\tilde{\boldsymbol{\xi}}) = \begin{cases} (\tilde{x}/20)^2 + (\tilde{y}/10)^2 = 1 & x \leq x^o \\ (\tilde{x}/20)^6 + (\tilde{y}/10)^2 = 1 & x > x^o \end{cases}$$

$$\textbf{Case 2: } \Gamma(\tilde{\boldsymbol{\xi}}) = \begin{cases} (\tilde{x}/12)^2 + (\tilde{y}/1.6)^2 = 1 & x \leq x^o,\ y \leq y^o \\ (\tilde{x}/32)^2 + (\tilde{y}/1.6)^2 = 1 & x > x^o,\ y \leq y^o \\ (\tilde{x}/32)^2 + (\tilde{y}/5.6)^2 = 1 & x > x^o,\ y > y^o \\ (\tilde{x}/12)^2 + (\tilde{y}/5.6)^2 = 1 & x \leq x^o,\ y > y^o \end{cases}$$

$$\textbf{Case 3: } \Gamma(\tilde{\boldsymbol{\xi}}) = \begin{cases} (\tilde{x}/12)^4 + (\tilde{y}/4)^2 = 1 & y \leq y^o \\ (\tilde{x}/12)^2 + (\tilde{y}/10)^2 = 1 & y > y^o \end{cases}$$

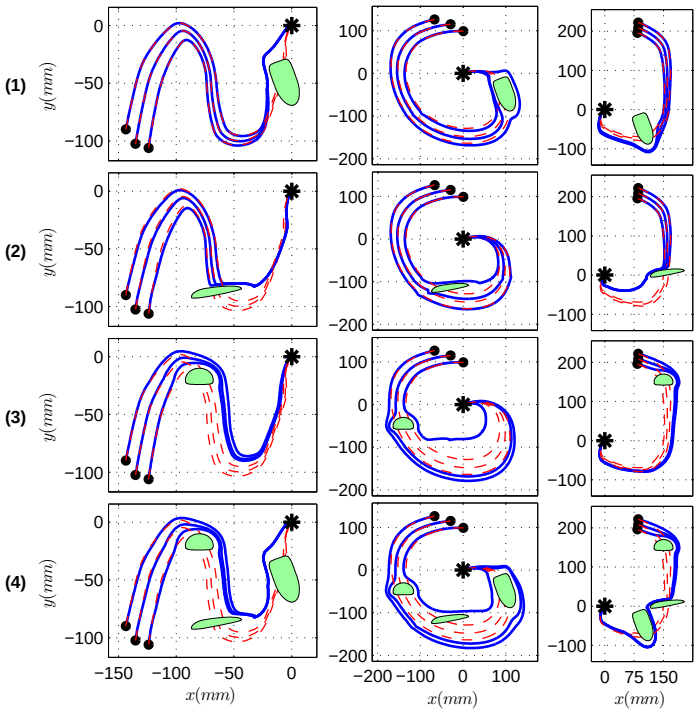
Case 4: Superposition of cases 1, 2, and 3

The obstacles in cases 1 and 2 are rotated by 110 and 10 degrees, respectively. We used the safety factor $\boldsymbol{\eta} = 1.3$ for all the obstacle models. For both autonomous and non-autonomous DS, the modified dynamics of the motions successfully reach the target without hitting the obstacles. Case 4 in Fig. 6.14 shows the result for the situation where multiple objects exist in the experiment.

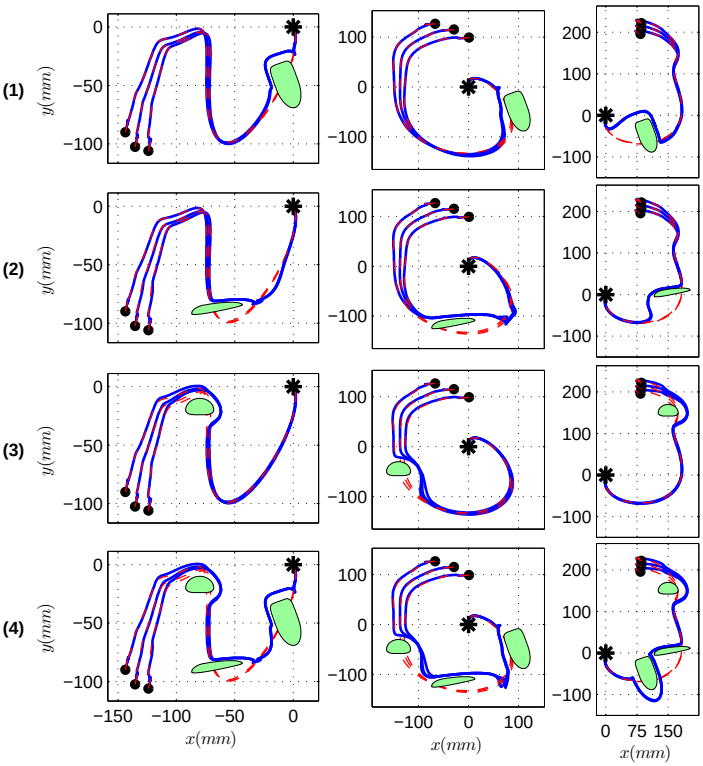
6.7.3 Robot Experiments

In this section we evaluate our obstacle avoidance method in six robot experiments (four in the Cartesian space and two in the robot joint space) performed on WAM and DLR arms. Depending on the experiment, the robot is kinematically controlled in either Cartesian or joint space. The controller command for the WAM and DLR arms are sent at 500 and 1000Hz, respectively. For the experiments in the Cartesian space, we use the damped least square pseudo-inverse kinematics to compute the robot's joint angles. The torque command to the robot is computed based on the desired kinematic command using an inverse dynamics controller. All the results illustrated in this section were recorded from the robot.

[9]The same principle can be used if the SEDS motions are modeled with a second or higher order DS.



(a) First order DS modeled by SEDS.



(b) Second order DS modeled by DMP.

Figure 6.14: Performance evaluation of the proposed approach on following three patterns in the presence of different obstacles. The motion patterns are modeled with two different approaches: SEDS and DMP. The initial and final points of the trajectories are indicated by solid circle and star, respectively. Please refer to the text for further information.

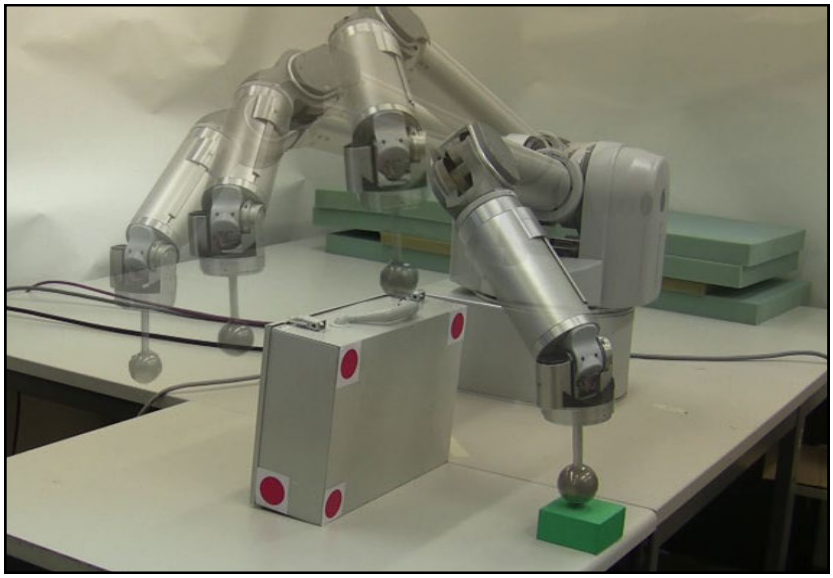
6.7.3.1 Experiments in the Cartesian Space on the WAM arm

The first experiment consisted of having the robot reach for an object while avoid hitting a table and a box. The height, length, and width of the table are 0.02, 3 and $3m$ respectively, and for the box these values are 0.24, 0.36, and $0.12m$. Note that we consider an extremely large value for the length and width of the table to limit all trajectories to the region above the table. The orientation and the position of the box are computed by detecting the four markers' location (blobs) placed on the box at the rate of 100 fps using two high-speed Mikrotron MK-1311 cameras. The position and orientation of the table are fixed and are given to the system.

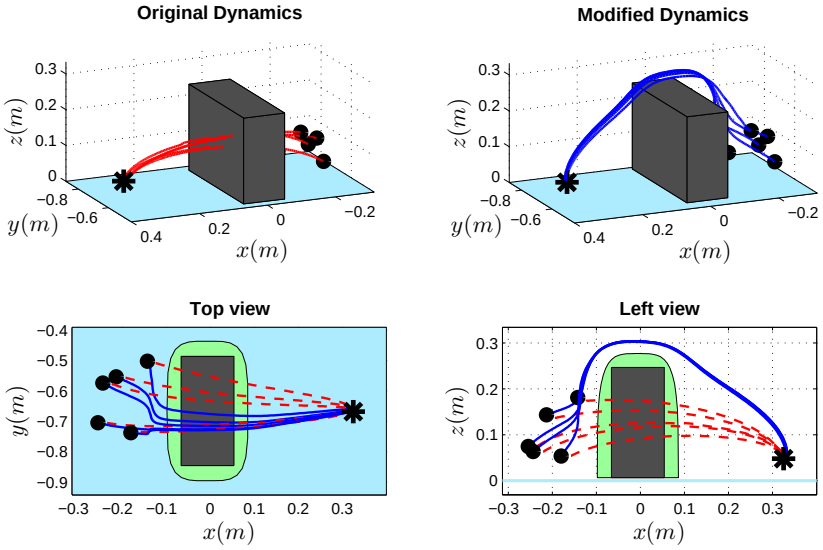
In this experiment, we define the motion in the Cartesian coordinates system. The original robot motion is learned using SEDS based on a set of demonstrations (in the absence of obstacles) provided by the user. Figure 6.15 represents the experiment set-up and the trajectories generated from the original and the modulated dynamics of the motion. As it is expected, all reproductions from the modified dynamics successfully avoid the box and reach the target. In this experiment, the box center is initially placed at $x^{c,B} = 0.0$, $y^{c,B} = -0.65$, and $z^{c,B} = 0.135$m with respect to the robot frame of reference. We define the box reference point to be at $x^{o,B} = x^{c,B}$, $y^{o,B} = y^{c,B}$, and $z^{o,B} = 0$, and use the analytical formulation $\Gamma(\tilde{\boldsymbol{\xi}})^B$: $((x-x^{o,B})/0.092)^4+((y-y^{o,B})/0.23)^4+((z-z^{o,B})/0.27)^4 = 1$ to model the box. The table is also modeled with $x^{o,T} = y^{o,T} = 0$, $z^{o,T} = -0.01cm$ and $\Gamma(\tilde{\boldsymbol{\xi}})^T$: $((x-x^{o,T})/3)^6 + ((y-y^{o,T})/3)^6 + ((z-z^{o,T})/0.01)^4 = 1$. We set the safety factor of the table and the box to $\boldsymbol{\eta} = 1.3$. For the box, we consider $\boldsymbol{\eta} = [2.5\ 1.5\ 1.2]^T$ to account for the large differences between the box height, length, and width.

Note that, though the box and the table are connected, we can avoid the problem highlighted in Fig. 6.8d by defining $z^{o,B} = 0$. In this way, the dynamic modulation matrix of the box always deforms trajectories towards its upper part. Thus no local minimum can be generated at the contact edges of the box and the table.

Adaptation to changes in the target position: To verify the adaptability of the system in a dynamic environment, we perform an experiment in which we continuously displace the target while the robot approaches it (see Fig. 6.16). During the reproduction, the position of the target is updated based on the output of the stereo vision system. Since the modulated dynamics preserves the asymptotic stability of the model, the system can adapt its motion on-the-fly to the change in the target position. Note that the instant adaptation to the target position is an inherent property of the SEDS modeling. In this experiment we are demonstrating the fact that our approach preserves all the properties of the SEDS model, while enabling it to perform obstacle avoidance.

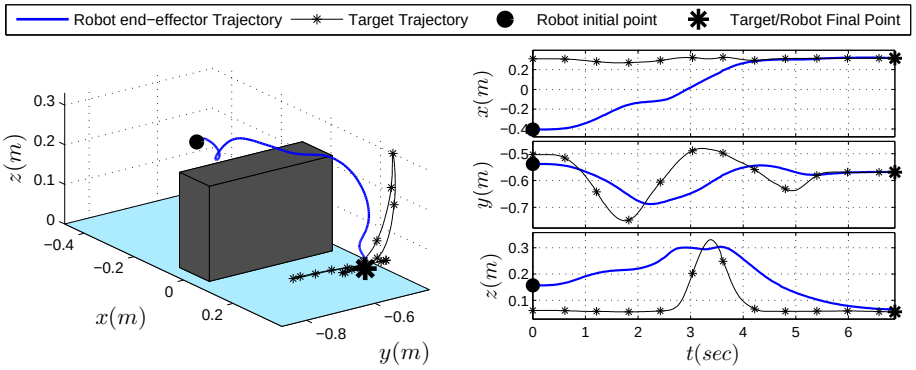


(a) The experiment set-up. The upper surface of the green block corresponds to the target point.



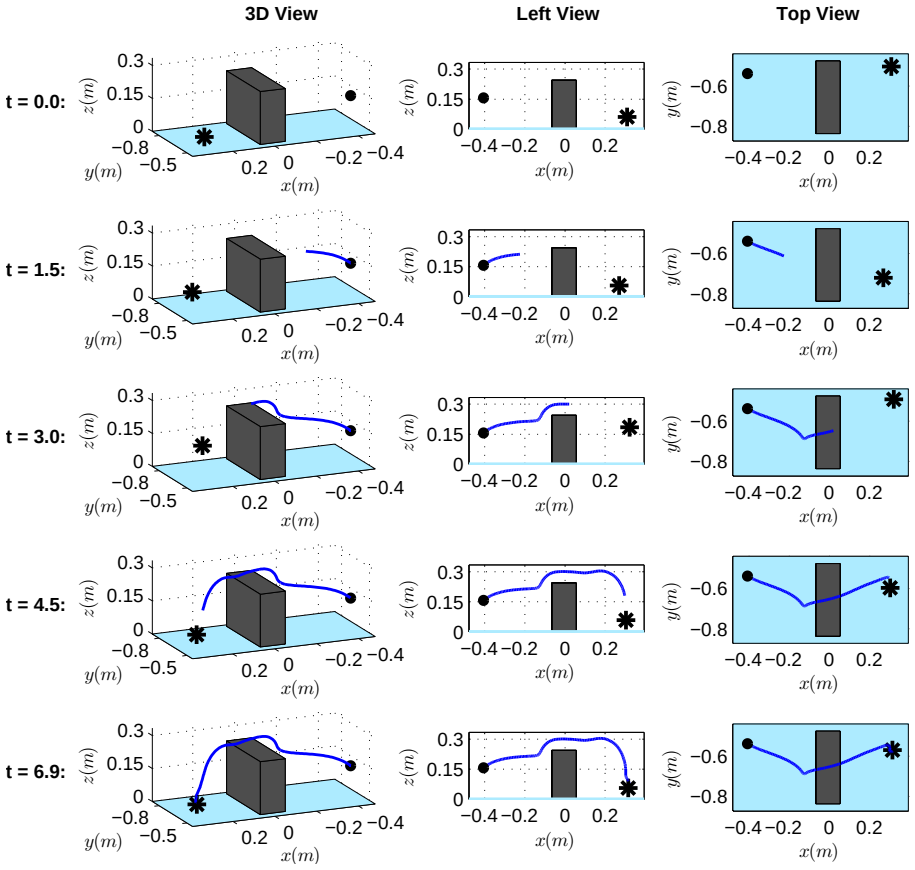
(b) Adaptation of the original dynamics of the reaching motion (**top-left**) with the dynamic modulation matrix (**top-right**). The graphes in the bottom row illustrate the top and left views of both dynamics.

Figure 6.15: Evaluation of our method in a static environment, where the WAM robot should reach for an object while avoid hitting the table and the box. Red dashed lines and solid blue lines correspond to the trajectories from the original and the modified dynamics, respectively. The black area represents the box outer surface, and the green area is its estimated analytical model. The light blue rectangle shows the upper surface of the table. The initial and final points of each trajectory are indicated by a solid circle and star, respectively.



(a) Illustration of the robot and target motions.

(b) Evolution of the robot and target motions along time.



(c) Illustration of the sequences of the motion.

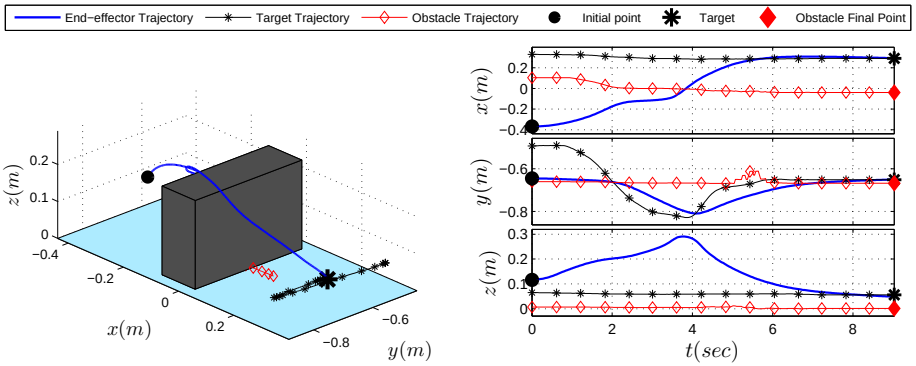
Figure 6.16: Adaptation of the model to the changes in the target position.

Adaptation to changes in both the target and obstacle positions: To evaluate the performance of the system in the presence of a moving obstacle, we extend the previous example to a case where both the target and the obstacle positions are changed as the robot approaches the target. Please note that for illustrative purpose, in this experiment we assume that the obstacle movement is "quasi-static". We will show later on in Section 6.7.3.2 an experiment where the obstacle's linear and angular velocity are taken into account during the collision avoidance.

Figure 6.17 demonstrates the obtained results. In this experiment, at the time between $t = 0$ and $t = 6$ seconds, the target is moved from its original position first in the opposite and then along the direction of the y-axis. The box also starts moving in the period between $t = 0$ and $t = 2$ seconds. During the reproduction, the target position and the box center and orientation are continuously updated based on the output of the stereo vision system. Similarly to the previous example, the system remains robust to these changes in the environment and successfully reaches the target.

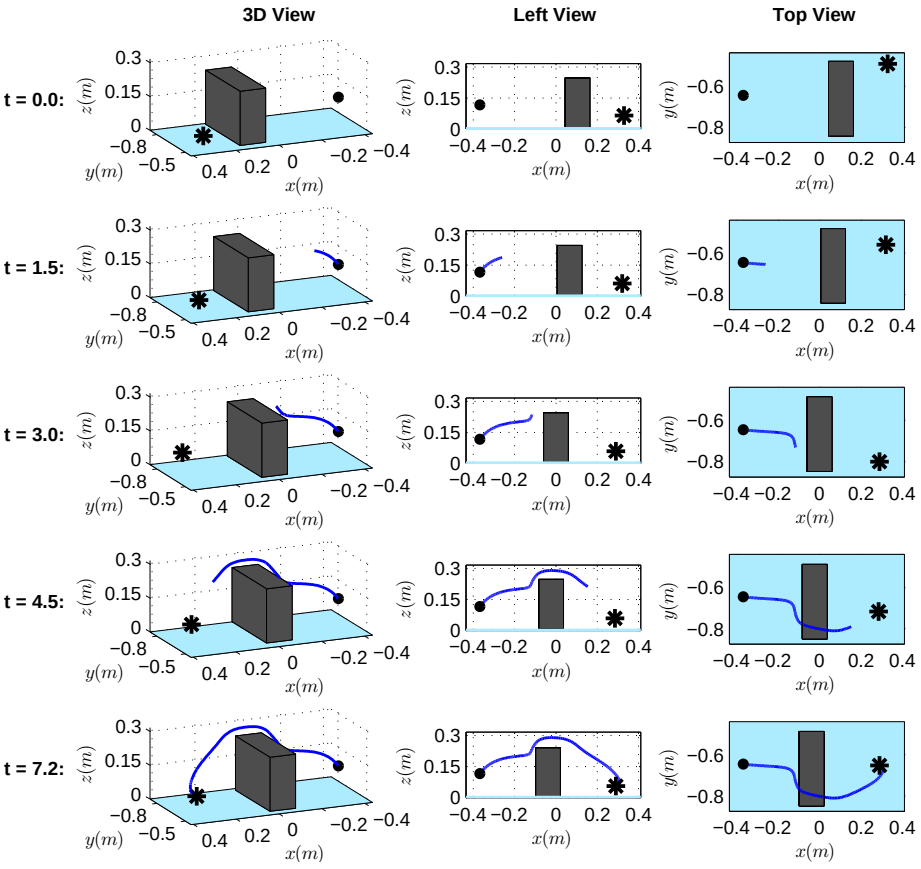
Evaluation in a more dynamic environment: We further evaluate our approach in a more dynamic environment where both the target and the obstacle are quickly displaced as the robot moves toward the target. Both positions of the target and the obstacle are detected at 100Hz. The obstacle is a ball with radius of $5cm$. We set its safety factor to $\boldsymbol{\eta} = 1.5$. Note that the safety factor of 1.5 results in a 2.5cm safety margin around the ball which is necessary to compensate for the size of the haptic ball attached to the robot's end-effector. Figure 6.18 shows the experiment set-up and the obtained results. The robot adapts on-the-fly its motion to both the obstacle and the target movement.

Evaluation in a complex environment: In this experiment we evaluate our method in the presence of several obstacles including a desk lamp, a pile of books, a Wall-E toy, a pencil sharpener, a book, a (red) glass, and a desk. The task consists of having the robot place a (transparent) glass on the desk, and in front of the person (see Fig. 6.19). The position and orientation of all the objects except the glass are pre-set. In order to have a more realistic experiment, at each trial we add a error vector $\boldsymbol{\epsilon}$ to the predefined position of each obstacle $\boldsymbol{\xi}^{o,k}$ to account for uncertainty in the environment, i.e. $\hat{\boldsymbol{\xi}}^{o,k} = \boldsymbol{\xi}^{o,k} + \boldsymbol{\epsilon}^{k}$. The value of each component of the error vector $\boldsymbol{\epsilon}^{k}$ is drawn from a Gaussian distribution with $\mathcal{N}(0, 0025)$. The position of the glass is actively tracked through the stereo camera described above. The maximum tracking error in sensing the glass position is ± 0.05m. The orientation of the glass is not measured, though it may change during each trial. We approximate all the obstacles with an ellipsoid envelope of the form $\sum_{i=1}^{3}(\tilde{\boldsymbol{\xi}}_i/\boldsymbol{a}_i)^{2\boldsymbol{p}_i} = 1$, where $\boldsymbol{a}_i > 0$ and $\boldsymbol{p}_i > 0$ are real and integer values, respectively. To compensate for the uncertainties, we consider a safety factor of $\boldsymbol{\eta} = 1.5$ for all the obstacles. The tail-effect of all the obstacles is removed (i.e. $\kappa = 0$), and the reactivity to the presence of the glass is increased by setting $\rho = 2$ (the default value of $\rho = 1$ is considered for other objects).



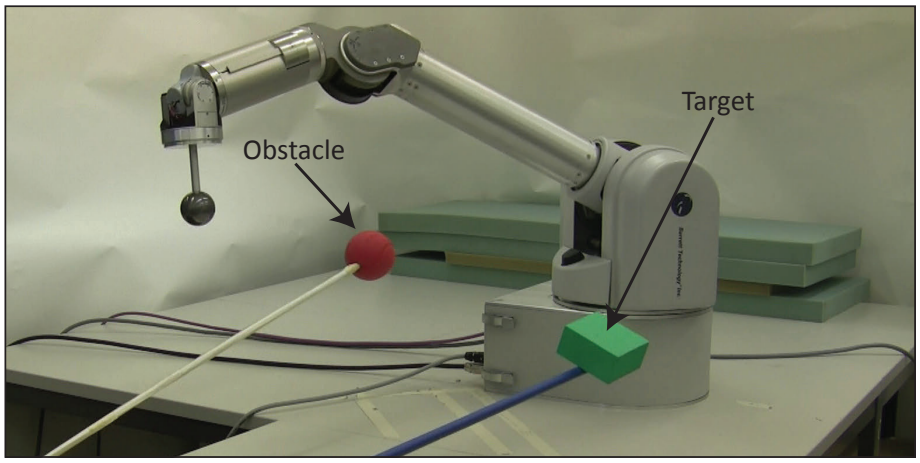
(a) Illustration of the robot, target and obstacle motions.

(b) Evolution of the robot, target and obstacle motions along time.

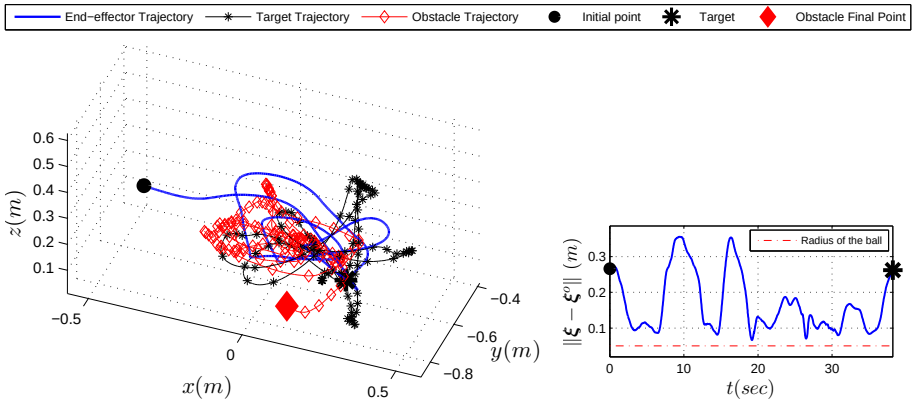


(c) Illustration of the sequences of the motion.

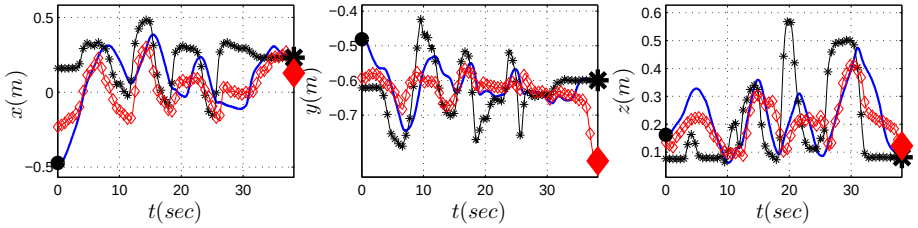
Figure 6.17: Robustness of the model to the changes in the target and obstacle positions.



(a) Experiment Set-up

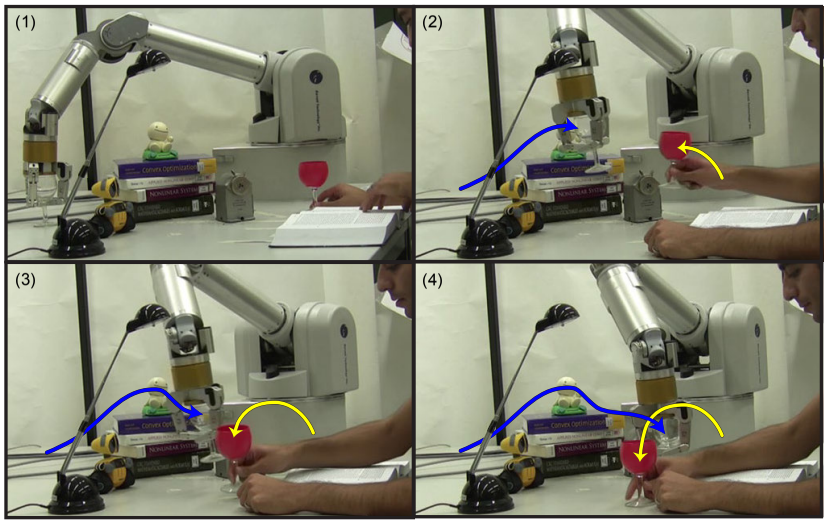


(b) Illustration of the robot, target and obstacle motions. **(c)** Distance to the obstacle.

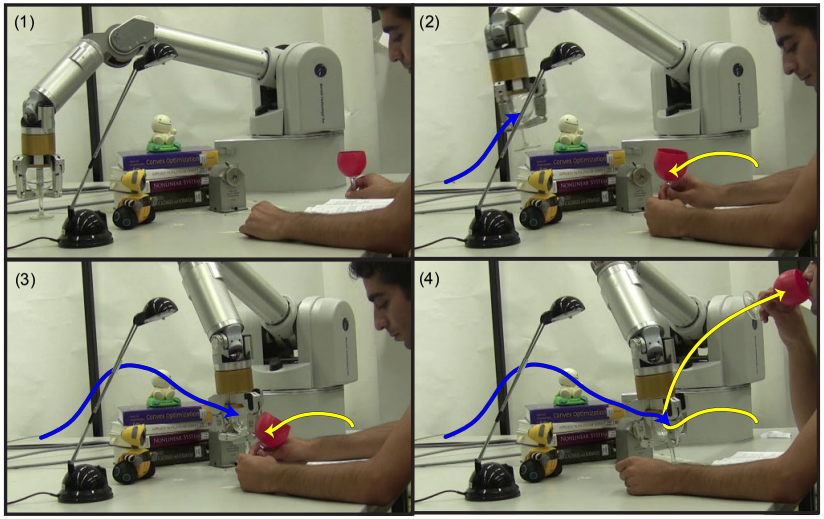


(d) Illustration of the evolution of the motion along time.

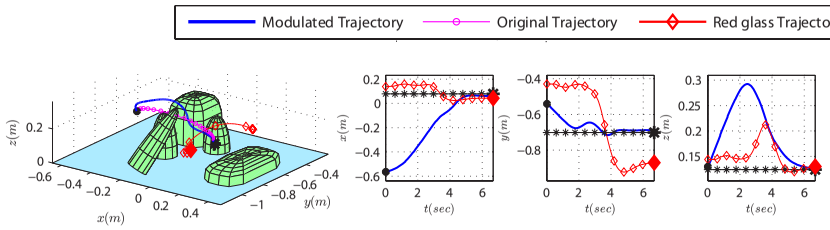
Figure 6.18: Validation of the proposed method in a dynamic environment, where both the target and the obstacle are displaced continuously. The obstacle is a ball with the radius of $5cm$. Please refer to the text for the further information.



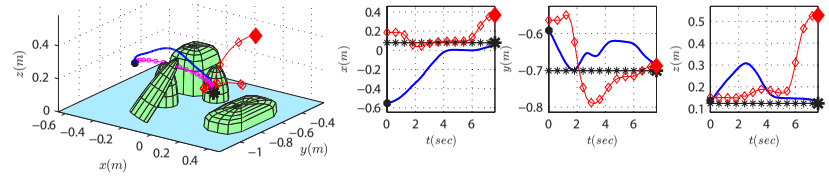
(a) Sequences of the motion for the first trial. In this experiment, the person moves the red glass from his right to his left hand side while the robot is approaching the target point. (1): The initial configuration. (2): The person intentionally moves the glass in a way that crosses the robot trajectory to the target point. (3) & (4): In order to avoid hitting the red glass, the robot deflects its trajectory towards the left side of the person, and then approaches the target from that side. The trajectories of the robot and the red glass are shown with blue and yellow curves, respectively. The traveled path is indicated with a solid line.



(b) Sequences of the motion for the second trial. (1) & (2): In this experiment, the person takes the glass from its right hand side and moves it to the target position while the robot is approaching. (3): In this situation, the robot stops near the red glass (and the target) since it cannot get any closer to the target. The robot waits at this position until the person clears the areas. (4): When the red glass is lifted, the robot moves towards the target point. The trajectories of the robot and the red glass are shown with blue and yellow curves, respectively.



(c) Illustration of the robot, target and obstacle motions for the first trial.



(d) Illustration of the robot, target and obstacle motions for the second trial.

Figure 6.19: Evaluation of the proposed method in a complex environment. In this experiment, the robot is required to put a glass on the desk and in front of the person, while avoid hitting several objects including a desk lamp, a pile of books, a Wall-E toy, a pencil sharpener, an open book, a (red) glass, and a desk. All the objects except the red glass are fixed and their convex envelope are shown in green. The trajectory of the red glass is indicated by red diamonds (for the clarity of the graph, we do not display the envelope of the red glass).

In this section, we report on two trials of this experiment, but we have also included two additional trials in the accompanying video. We use the same DS function that was described in the previous robot experiments to control the robot motions. In the first trial, the person moves the red glass from his right to his left hand side (i.e. along the negative direction of the y-axis) while the robot is approaching the target point. The person intentionally moves the glass in a way that crosses the robot trajectory to the target point (see Fig. 6.19a). In order to avoid hitting the red glass, the robot deflects its trajectory towards the negative direction of y-axis, and then approaches the target from its left side (in Fig. 6.19c, see the robot trajectory along y-axis in the time period $t = [3\ \ 4]$ seconds).

In the second trial, the person takes the glass from its right hand side and moves it to the target position while the robot is approaching (see Fig. 6.19b). In this situation, the robot stops near the red glass (and the target) since it cannot get any closer to the target (in Fig. 6.19d, see the time evolution of the robot trajectory in the time period $t = [4\ \ 6]$ seconds). The robot waits at this position until the person clears the areas. When the red glass is lifted, the robot moves towards the target point.

6.7.3.2 Experiments in the Cartesian Space on the DLR arm

In this section we evaluate our approach in the presence of a fast moving obstacle, where the quasi static-assumption is no longer valid. The experiment consisted of having the 7-DoF KUKA DLR arm stay in a default target position while a box is slid towards the robot at high speed. Thus the robot should react quickly and change its position so that the box passes without any collision (see Fig. 6.20).

The KUKA robot is controlled in the Cartesian coordinate system, and the control commands are sent at 1000Hz. A SEDS model is used to control the robot motion by generating velocity commands to keep the robot's end-effector close or, when it is feasible, at the target point. We define the box reference point at $x^{o,B} = x^{c,B}$, $y^{o,B} = y^{c,B}$, and $z^{o,B} = 0$, and model it with the analytical formulation $\Gamma(\tilde{\boldsymbol{\xi}})^B$: $((x-x^{o,B})/0.055)^2+((y-y^{o,B})/0.165)^2+((z-z^{o,B})/0.23)^4 = 1$. Other parameters are set as follows: $\boldsymbol{\eta} = [3.5\ 2.0\ 1.5]^T$, $\rho = 2$, $\sigma = 30$, and $\kappa = 0$. The box's position and orientation are tracked at 240Hz using an OptiTrack vision system. We use a Kalman filter to reduce the noise effect on estimations. The working table is defined similarly to Section 6.7.3.1, and its position is set fixed in the whole experiment.

In total we ran 20 trials, lasting between 0.8 to 1.3 seconds, in which the box was slid from different initial configurations with various linear and angular velocities. In each trial, the box was set to an initial distance of about 0.5 meter away from the robot and was thrusted so as to reach a maximum linear velocity of 0.6 $\sim$ 1.5 m/s and/or a maximum angular velocity of 40 $\sim$ 120

deg/s. In 16 out of the 20 trials, the robot successfully managed to dodge the box. Figure 6.20 shows sequences of the motion for four of the trials. The trajectories of the robot's end-effector and the box, and the magnitude of the box's linear and angular velocities are also illustrated in Fig. 6.21.

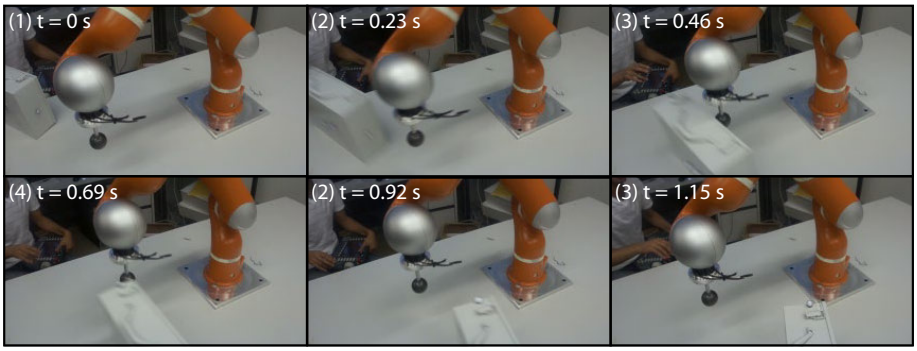
The four failure cases could possibly be due to two factors that are not currently considered in our implementation: 1) The filtering of the object's position and orientation introduces a lag in determining the current linear and angular velocities of the box. In situations where the box is moving and rotating fast at a very close distance to the robot, the presence of this lag could yield collision with the obstacle. 2) The robot's joints cannot move faster than a certain value due to the hardware limitation, and hence collision with the obstacle is inevitable. Figure 6.22 shows the sequences of the motion for one of the failure cases. In this trial, though the avoidance seems successful at the initial stage of the motion, the box hit the end-effector from the backside due to the wrong estimation of the object's angular velocity.

The first factor can be alleviated by using a more advanced filter or by increasing the safety factor. However, the second case cannot be easily tackled. Some improvements might be achieved by using a planner technique that could take into account such hardware limitations during the path generation. However, as in the above failure situations the obstacle is moving fast at a very close distance to the robot, this planner should be extremely fast to provide a valid solution within an order of millisecond (recall the robot is controlled at 1000Hz).

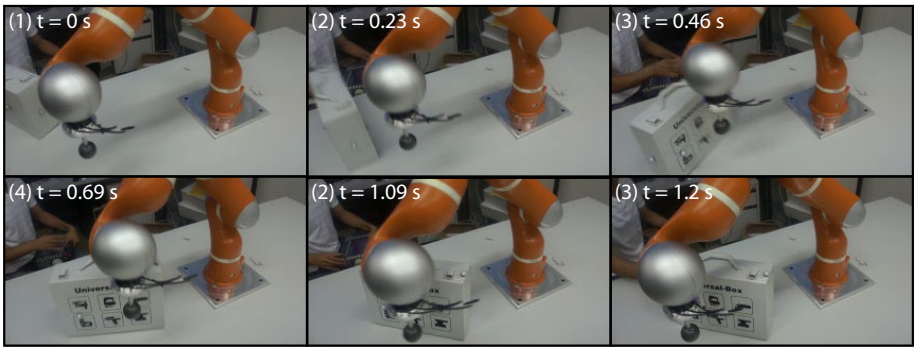
6.7.3.3 Experiments in the Joint Space on the WAM arm

In this section, we validate our approach in $d = 7$ dimensions, by controlling this time the WAM arm's 7 joints, i.e. $\boldsymbol{\xi} = [\boldsymbol{\theta}_i]$, $i = 1..d$. In the first experiment, we use our obstacle avoidance approach to limit the movement range in the second joint of the robot to values below -65 degrees. To reach this goal, we define a 7-dimensional obstacle $\Gamma(\boldsymbol{\theta}) = \sum_{i=1}^{7}((\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^o)/\boldsymbol{a}_i)^4$ with $\boldsymbol{a} = [500; 2; 500; 500; 500; 500; 500]$, $\boldsymbol{\theta}^o = [0; -63; 0; 0; 0; 0; 0]$, the safety factor $\boldsymbol{\eta} = 1.2$, and the reactivity $\rho = 5$. The original DS is defined in the joint space and is learned based on a set of demonstrations in the robot joint space using the SEDS learning algorithm. Figure 6.23 illustrates the generated trajectories from the original and the modified dynamics. As it is expected, in the modified dynamics, the robot successfully reaches the target while the value of the second joint remains below the desired value.

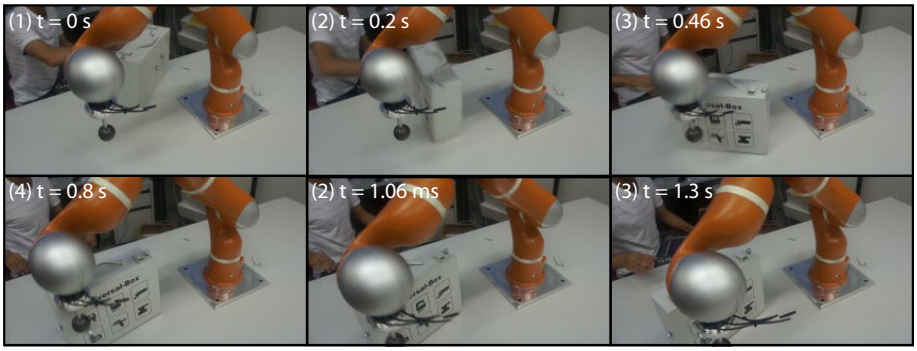
In the second experiment, we use our approach to avoid two 7D spherical obstacles defined in the robot joint space. The original robot motion is a cyclic movement in $\boldsymbol{\theta}_1$-$\boldsymbol{\theta}_2$ plane with $\dot{\boldsymbol{\theta}}_1 = \boldsymbol{\theta}_2$ and $\dot{\boldsymbol{\theta}}_2 = -\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2(1-(\boldsymbol{\theta}_1/5)^2)$ and $\dot{\boldsymbol{\theta}}_i = 0$, $\forall i \in 3..7$. The obstacles have radius of $r^{o,1} = r^{o,2} = 5$ degrees and are placed in $\boldsymbol{\theta}^{o,1} = [-100; 45; 1; 61; 1; -29; 1]$ and $\boldsymbol{\theta}^{o,2} = [-80; 45; -1; 59; -1; -31; -1]$, respectively. The safety factor of $\boldsymbol{\eta} = 1.2$ is used in this experiment.



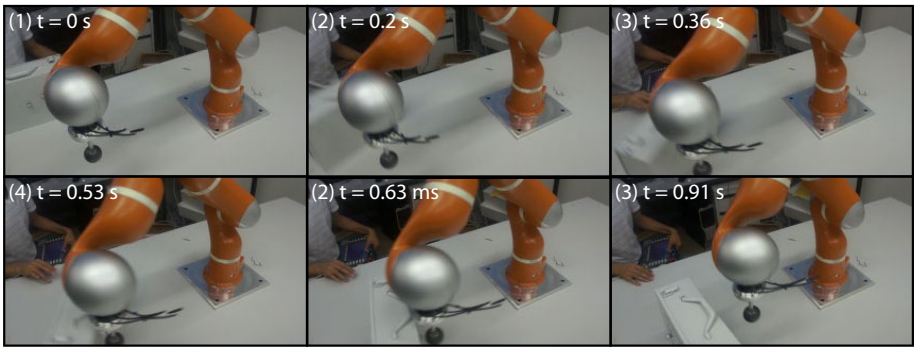
(a) First trial.



(b) Second trial.

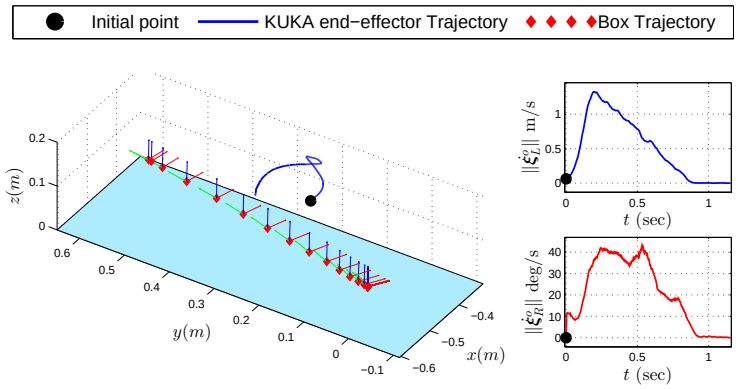


(c) Third trial.

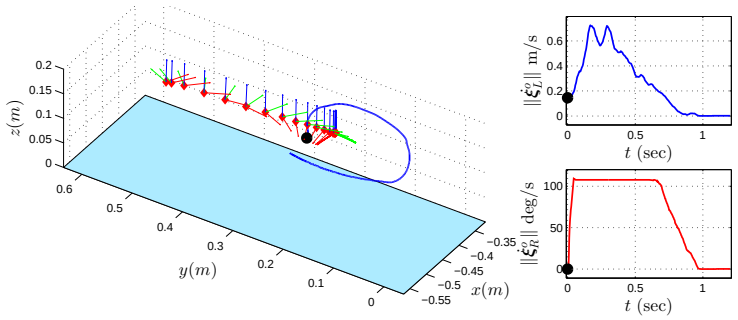


(d) Forth trial.

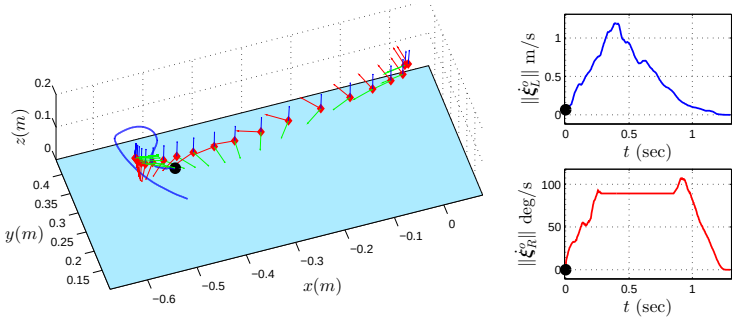
Figure 6.20: Illustration of sequences of motion for 4 out of the 20 executed trials. In this experiment the robot was required to dodge a sliding box that was launched 20 times from different initial configurations with various linear and angular velocities. For further information please refer to Section 6.7.3.2.



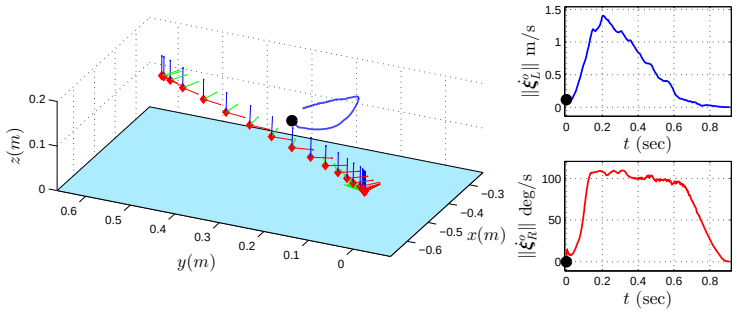
(a) First trial.



(b) Second trial.



(c) Third trial.



(d) Forth trial.

Figure 6.21: Illustration of trajectories of the robot's end-effector and the box, and the magnitude of the box's linear and angular velocities for the four trials shown in Fig. 6.20. In these graphs, the x, y, and z axes of the box's frame of reference are shown with red, green, and blue vectors, respectively. For further information please refer to Section 6.7.3.2.

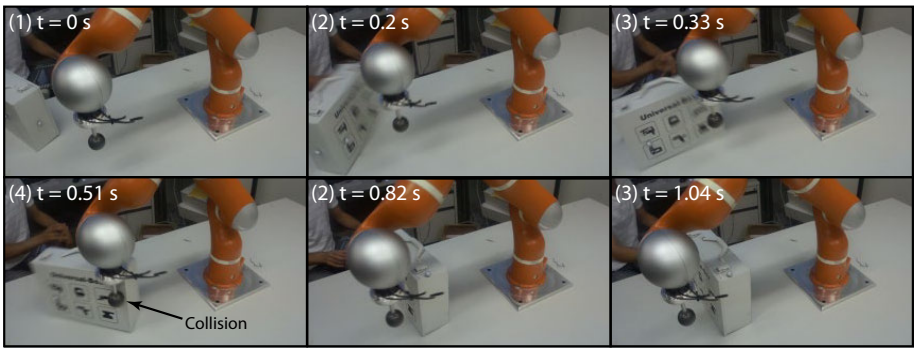
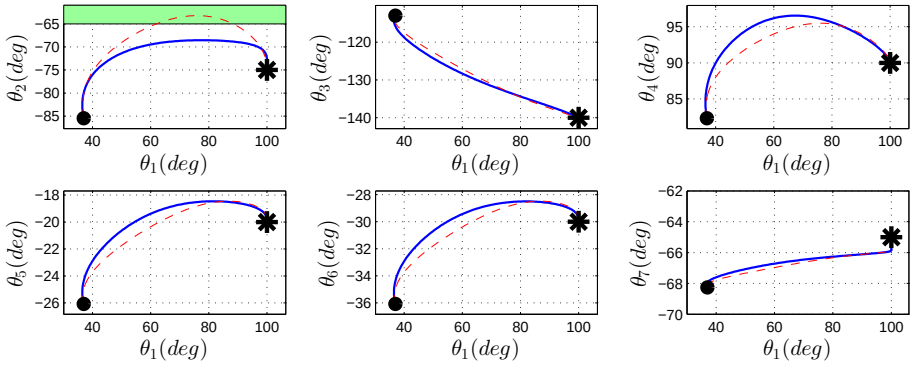
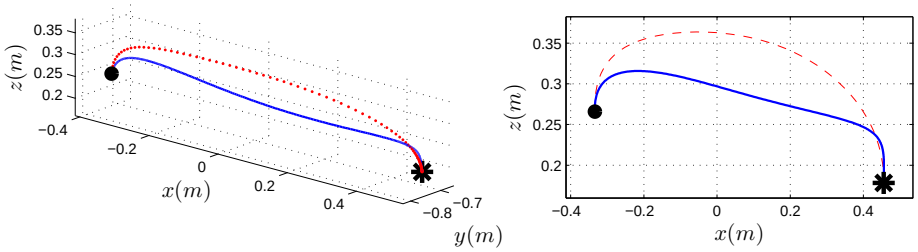


Figure 6.22: Illustration of sequences of motion for one of the four cases in which the robot failed to successfully dodge the box.



(a) Robot trajectories in the joint space.



(b) Illustration of the robot movement in the robot task space.

Figure 6.23: Using the proposed obstacle avoidance module to limit the movement range in the second joint of the robot to values below −65 deg. The red dashed line and the blue solid line corresponds to the trajectories generated by the original and the modified dynamics, respectively. The obstacle is shown in green. The initial and final points of the motion are indicated by a solid circle and star, respectively.

Fig. 6.24a illustrates the evolution of the motion in the absence and presence of the obstacles. One can observe that the modulated dynamics deviates in the presence of obstacles, and due to the induced coupling via the dynamic modulation matrix[10], the robot also starts showing cyclic behavior in previously static joints, i.e. $\boldsymbol{\theta}_i$, $i = 3..7$. Figure 6.24b shows the distance to the closest obstacle along the time. Here, one can observe that while the original motion penetrates into the obstacle, the modulated dynamics can smoothly avoid the obstacles. The corresponding robot motion in the task space is shown in Fig. 6.24c. Note that this work does not claim that the cyclic behavior is always preserved in the presence of the obstacles.

6.8 Summary and Conclusion

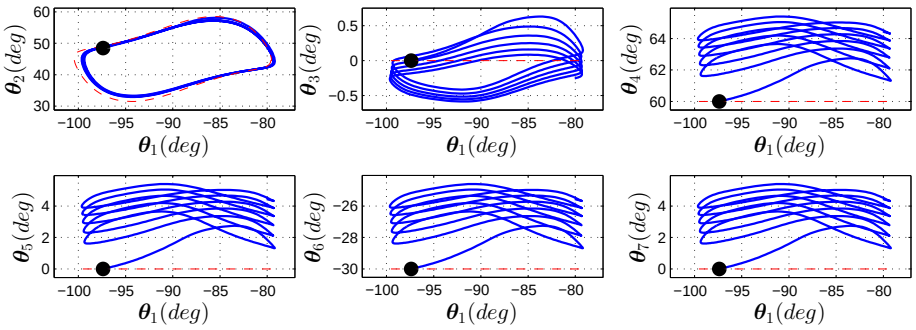
In this chapter, we proposed a DS-based approach to realtime obstacle avoidance for a case where robot motions are given by autonomous or non-autonomous DS, and the obstacle(s) are convex. The method is derived for a d-dimensional DS, hence can be used in both the Cartesian and configuration spaces. The proposed method can handle multiple obstacles, and do not modify the equilibrium points of the original dynamics. However, in the presence of obstacle(s) the method may lead to the appearance of saddle points and local minima along the obstacles' boundaries. These points can be tackled through Algorithm 6.1.

The presented approach requires a global model of the environment and an analytical modeling of the obstacles boundary. When the analytical description of the obstacle is available, our method guarantees that all obstacles will be avoided safely. However, the analytical equation of the obstacle or its accurate status (i.e. position and orientation) may not be available all the time. To generate the analytical equation, it is possible to use one of the state-of-the-art bounding-volume algorithms (e.g. Benallegue et al. (2009); Lahanas et al. (2000); Welzl (1991)) to approximate a convex BV on the output of the vision system. In the worst case when there is little time to generate the bounding volume, one could quickly fit the point cloud with an ellipsoid.

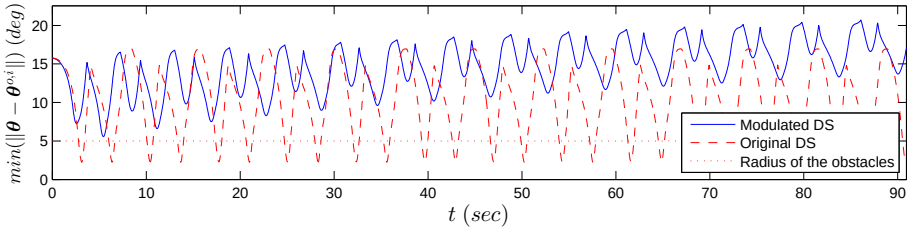
The presented algorithm is able to cope with uncertainty in the obstacle's position by allowing certain safety margins around the obstacle. The larger the safety margin, the more robust the system is to uncertainty in the obstacle position. Note that in the presence of an unforeseen object, uncertainty in the obstacle's position, or hardware limitations, our algorithm no longer guarantees the safe avoidance of the obstacle, and can only strive for the best performance.

All theorems derived in this work are based on the continuous state space assumption; however, in real experiments, robot motions are usually generated

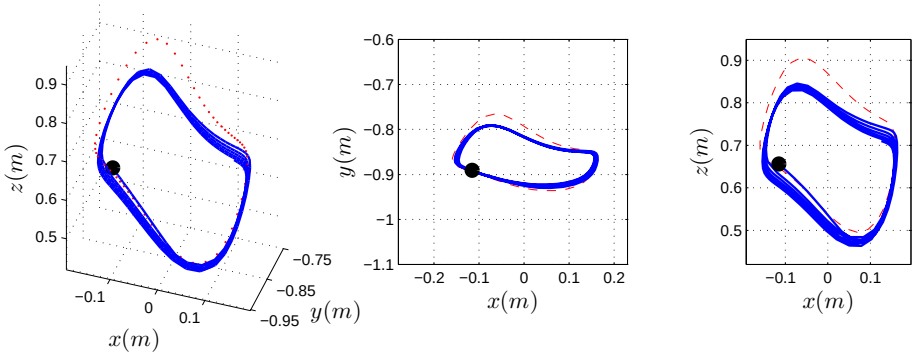
[10]Note that the motions across $\boldsymbol{\theta}_i$, $i = 3..7$ would become uncoupled if the obstacles were placed at $\boldsymbol{\theta}^{o,1} = [-100; 45; 0; 60; 0; -30; 0]$ and $\boldsymbol{\theta}^{o,2} = [-80; 45; 0; 60; 0; -30; 0]$.



(a) Robot trajectories in the joint space. The solid black circle indicates the starting point of the motion.



(b) The distance to the closest obstacle.



(c) The corresponding robot motion in the task space.

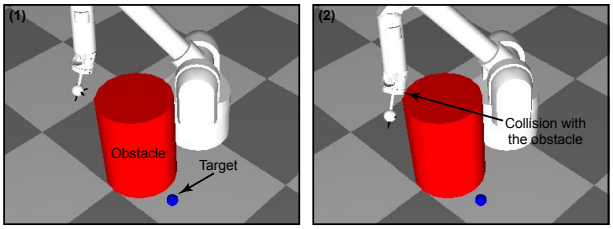
Figure 6.24: Illustration of applying the obstacle avoidance module in the robot joint space. In this figure, the red dashed line shows the original cyclic motion and the solid line demonstrates the modulated motion in the presence of two 7D spherical obstacles with the radius of 5 degrees. The robot motion is defined in the joint space and its evolution is shown in **(a)**. Please refer to the text for the further information.

with a finite number of points (discrete modeling). Thus the choice of integration time step is important specially in the close vicinity of the object. In fact, when a big integration time step is used, for trajectories that are very close to an obstacle, it is very likely that the subsequent point falls inside the obstacle due to the integration error. In this situation, trajectories tend to remain inside the obstacle (because the boundaries are impenetrable, no trajectory can enter or leave the obstacle). In our experiments, we did not face such an issue by considering the integration time steps of 0.01 and 0.002 sec in all simulations and robot experiments, respectively.

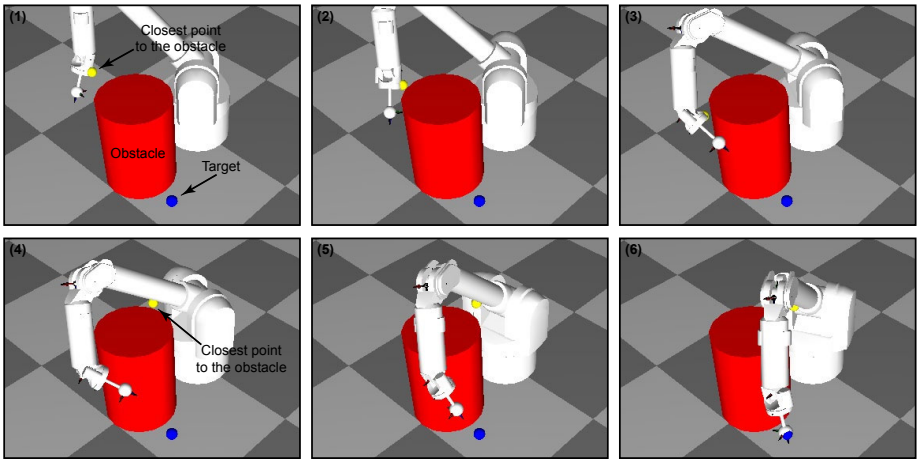
The presented work is limited in that it can only be applied to convex shaped obstacles. While Theorem 6.2 still holds for concave shape, the simple Algorithm 6.1 to overcome local minima on the boundary can no longer apply and an alternative solution must be sought.

The presented work considers obstacle avoidance for a point robot. However, it is also possible to integrate other algorithms to perform collision avoidance for the whole robot. For example, while the end-effector follows the commanded velocity from the proposed approach, one can use the kinematics null-space to avoid link collision (Maciejewski & Klein, 1985). An example of such an extension was implemented by Burak Zeydan as a part of his semester project conducted under my supervision at LASA (see Appendix F for the project definition). This work uses the proposed obstacle avoidance approach to guide the robot's end-effector, and simultaneously determines the closest point on the robot to the obstacle(s). This point is then driven away using the remaining degrees of freedom. Figure 6.25 shows an example of using the above procedure to perform the whole body collision avoidance on the simulator of the Barrett WAM arm. The simulated environment is provided by RobotToolKit[11]. It should be noted that this approach is, however, subject to local minima and convergence to the target may no longer be ensured.

[11] RobotToolKit is an open-source software for simulation and real time control of robotic systems. This software was developed by Eric Sauser at LASA, EPFL



(a) In this example, even though the end-effector can successfully avoid hitting the obstacle, it is not enough to safely avoid the collision of the whole arm with the obstacle.



(b) In this example, the robot's end-effector follows the proposed obstacle avoidance scheme, and at each iteration the closest point on the robot to the obstacle (marked with the green sphere) is computed and driven away thanks to the redundant degrees of freedom. As the robot moves, the closest point to the obstacle may slide on the same limb or jump to another limb.

Figure 6.25: An example illustrating the whole body collision avoidance that uses the presented obstacle avoidance scheme to control the end-effector's motion, and the method described in (Maciejewski & Klein, 1985) to extend it to the whole body collision avoidance.

Chapter 7

Conclusion

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing (1912-1954)

The work we have presented in this thesis opens new interesting doors in the fields of machine learning and robotics. In this chapter, we provide a brief summary of the major contributions of this work, and bring to light its main limitations along with possible directions of improvement.

7.1 Main Contributions

The main contribution of this dissertation lies in providing a generic and unified framework based on DS, capable of generating various robot discrete movements ranging from simple pick and place motions to agile striking movements. The learning algorithms presented in this thesis can build an estimate of nonlinear multi-dimensional DS from a set of examples while ensuring its global or local asymptotic stability at the target. As outlined in Chapter 3, to date, existing DS-based approaches to encode robot motions rely either on some heuristics with the aim to build a locally stable estimate of nonlinear DS without any guarantee that such a model is feasible, or they depend on a (time-dependent) switching mechanism to ensure stability by switching from an unstable nonlinear DS to a stable linear DS. This was the first time that a statistical-based learning algorithm was suggested which can actually ensure global stability of nonlinear DS during the training phase.

In this thesis, we have also introduced a DS-based obstacle avoidance approach that can be integrated into the above framework in order to provide a useful control policies when multiple static and moving objects exist in the robot's workspace. The proposed approach has a level of reactivity similar to existing local obstacle avoidance methods, while it ensures convergence to

the target proper to global obstacle avoidance techniques. As outlined before, our contribution to obstacle avoidance is not intended to devise a new concept that outperforms the existing approaches. Instead we aimed at providing a technique that can seamlessly integrate into the framework described above, without compromising its features such as convergence to the target, adaptability and robustness, reactivity, applicability to different models, etc.

The prominent features of the presented DS-based framework can be summarized as follows: 1) it allows a naive user to program robots to perform discrete movements using a natural means of demonstration, namely kinesthetic teaching, 2) it generates robot motions that are inherently robust to perturbations, and can instantly adapt to new situations in a dynamically changing environment, 3) it provides a means to perform collision avoidance in the presence of multiple static and moving obstacles, and most importantly at the kinematic level 4) it guarantees convergence of all trajectories to the target.

7.2 Limitations and Future Work

The limitations and drawbacks of the presented framework have already been discussed in their corresponding chapters. In this section, we elaborate more on some of the important limitations and provide some possible research directions that can stem from the work conducted in this thesis.

Choice of Kinematic Representation

Throughout this thesis, we have defined robot motions at the kinematic level, and have assumed that there is a low level tracking controller that converts kinematic variables into motor commands. There is a limitation inherent to this assumption: the dynamics of the robot as well as its hardware limitations are not explicitly taken into account during motion generations with our approach. Despite the facts that 1) the robot's hardware limitations are implicitly considered through learning from demonstrations, and 2) the DS model can partly compensate for deviations from the desired trajectory due to hardware limitations by instantly adapting a new trajectory, there is yet no theoretical proof that the whole system is capable of performing all the motions that are generated from the learned DS.

The above concern is less problematic in fully actuated robotic systems as compensation by DS is most of the time enough to tackle hardware limitations (at the cost of executing the motion at a slower pace). However, this issue is more critical when working with under-actuated robots as it may not be feasible to control these robots in some parts of the state space. Hence, more in-depth analysis and evaluations should be performed on this issue.

For fully actuated robots, one possible way to tackle the hardware limitations such as velocity limits is to explicitly consider them as constraints of the learning

techniques. Joints limits can also be formulated as the optimization's constraints if the task is defined in the $\mathcal{C}$-space. Considering joint limits as constraints for the tasks that are defined in the operational space is non-trivial and require further investigations.

Another possible way that could work for both under-actuated and fully actuated robotic systems is to leverage on the notion of funnels and define the whole task as a chain of connected funnels. The motion in each funnel can be modeled with a DS, which acts locally within its associated region. The advantage of using such modeling is that it could simplify the problem by estimating the nonlinear dynamics of the robot with a simpler model (e.g. a linearized model). Hence, the verification of the hardware limitations could become more tractable.

Necessity of Following Demonstrations

Throughout this thesis we have assumed that a task's demonstrations are consistent according to Eq. (4.1), and thus variations in the demonstrations (that passes through the same point) are only due to the noise. Based on this assumption, we have presented different learning algorithms to build a DS model of the motion so as to follow the demonstrations as accurately as possible. Despite showing the validity of this assumption in many robot tasks, there could be some scenarios where the variations in the demonstrations are not only due to the noise but also to the task null space movements. For these scenarios, a control policy that accurately follows the observed demonstrations could be inappropriate. Thus, to obtain a useful control policy, one should extract the null space component from the demonstrations prior to learning the DS model.

Null space movements could be due to different reasons such as the task's unknown constraints. Depending on the task complexity, null space component extraction could be very difficult and may only be feasible in particular cases. The work by Howard et al. (2009) shows some interesting results along this direction in which unconstrained control policies can be learned from demonstrations that are subject to a specific class of constraints.

Besides to the cases with task null space movements, there are also other scenarios where it may not be desirable to accurately follow the user demonstrations. This could be motivated by the fact that different dynamics may require following different trajectories to achieve the same final result, the so-called correspondence problem (Dautenhahn & Nehaniv, 2002). Throughout this thesis we have avoided addressing the correspondence problem by demonstrating motions from the robot's point of view, i.e. by passively guiding the robot's arm through the task. However, in case kinesthetic demonstrations are not possible and thus demonstrations are collected from another agent with different dynamics, further investigations should be done in order to obtain an appropriate mapping between the movements of the demonstrator and the apprentice.

Timely execution of motions

In the framework presented in this thesis, the timely execution of motions is not explicitly encoded in the DS. Thus, our approach at its present form cannot be used in tasks such as playing tennis or catching a flying object where timing becomes crucial (the robot should be at a certain location, at a certain moment). A possible way to encode this feature into our framework is to multiply the output of the DS by a modulation factor. This factor is similar to the strength factor that is presented in Section 5.2 with the only difference that its value now varies with time. In this formalism, the modulation factor should be actively updated based on the expected and the estimated time-to-reach to the target so that the robot reaches to the rendezvous point at the desired time. Such extension is now an ongoing research of other PhD students at LASA, and interesting results have been obtained for catching flying objects such as a half-filled bottle of water (S. Kim et al., 2010; S. Kim & Billard, 2012).

Online Learning

As outlined before, online learning is often crucial to allow the user to refine the model in an interactive manner. In Section 4.6, we have presented the SEDS-II learning algorithm that allows online learning of DS models through the use of LWPR (or other possible regression techniques that support online learning). However, in our approach the online learning is only at the level of the estimation of DS, and thus the new updates through online learning might be ignored after applying the stabilizing command. In other words, the result from online learning is only valid if it is in accordance with the estimated metric of stability, which is currently learned offline. In most cases, the above limitation is not critical as there are some flexibilities in SEDS-II, which allows the output from the DS to form an angle between $-\pi/2$ to $\pi/2$ with the gradient of the energy function. Thus, as long as the modifications through online learning do not fundamentally change the global features of the motion, it is very likely that the original energy function would not impose any limitation.

Nevertheless, in cases where the above assumption is not valid, a trivial solution is to retrain the energy function with both the old and new datasets. Although learning of the energy function is fast (in our experiment it was on average within a few seconds), this solution is not very elegant as 1) it requires keeping all the training data points which could yield to some data storage problem, and more importantly, 2) it could impose some notable discontinuities in the robot behavior right before and after the training. The latter is due to the fact that in case of using the WSAQF parameterization, the optimization may converge to different locally optimal solutions at each retraining. As a result, in order to have a full online learning support, further investigations should be conducted to allow continuous refinement of the energy function along with the online modification of the original DS.

The notion of Coupled DS

Throughout this thesis, we have shown examples of motions that were performed on arm manipulators with at most 7 degrees of freedom. Now assume a more complex robotic system, for example a humanoid, with tens of degrees of freedom. In such systems, although it is theoretically possible to model the motions for all the degrees of freedom with a single DS, in practice, it is non-trivial (if not impossible) to provide sufficient demonstrations to cover such a high-dimensional state space. Apart from being practically difficult, such modeling could be very inefficient as the task may not require considering correlation across all axes in the state space.

An interesting extension to this work is to split the degrees of freedom of the robot into a set of meaningful submanifolds (e.g. left arm, right arm, left fingers, right fingers, and so on) and then learn the motion of each submanifold (e.g. each limb) separately from their respective demonstrations. As successful execution of a task may require proper coordination between all these submanifolds, a set of coupling terms should be also considered to model spatial correlations between these submanifolds without compromising the global stability of the whole system.

The above extension is currently an ongoing research of other PhD students at LASA, and preliminary results have been obtained for reach-to-grasp motions (Shukla & Billard, 2012b). In this work, the hand and the fingers are driven with two separate SEDS models. The introduction of the coupling term between these two DS models allows the robot to seamlessly and rapidly adapt the finger motion in coordination with the hand postures.

Multi-attractors DS

Another natural extension to the single attractor DS that is developed in this thesis is to have different attractor topologies in the form of multiple-discrete attractors and continuous attracting surfaces. Such extension have a direct application in grasping complex objects where several grasping postures may exist. Depending on the current state of the robot, one may prefer one of these postures (attractors) to others. Modeling all these attractors in a single DS is advantageous as it would enable realtime switching between attractors in the case of perturbations. Current work by another PhD student at LASA studies this approach using an augmented-SVM model to partition the region of attraction of each target point. At each partition, a SEDS model is used to derive the motions, and further constraints are derived to ensure that the generated trajectories from SEDS models do not cross the boundaries and remains within the partition of their corresponding attractors, see (Shukla & Billard, 2012a) for further details.

Whole body Obstacle avoidance

The presented work considers obstacle avoidance for a point robot. Though a trivial solution for performing the whole body collision avoidance can be achieved by exploiting the kinematic null-space in case of redundant manipulators (see Appendix F for the project definition), global convergence to the target can no longer be ensured. Although there are other techniques to avoid link collision (e.g. the elastic band approach), to the best of our knowledge, there is yet no DS-based obstacle avoidance technique with this feature. Due to the promising properties of DS-based approaches, it would be an interesting research direction to extend the presented approach for the whole body collision avoidance while ensuring the global convergence to the target (if the target is reachable).

Learning of Hitting parameters

In Chapter 5, we have extended our formulations to perform hitting motions. We have evaluated this approach in the context of playing minigolf on a flat field, and have provided a possible mechanism for its adaptation to hit the ball at a desired speed and direction. As outlined in Section 5.1, performing hitting motions in tasks such as minigolf requires two parts: 1) a basic hitting motion model, 2) a set of valid hitting parameters. While learning of the former has been covered in this thesis, further work should be done along the latter for fulfilment of a task's requirement (for example sinking the ball in minigolf).

A preliminary study of this question was conducted as a master thesis directed under my supervision. In this study, we proposed and compared two statistical methods, GMR and GPR, to learn a model of hitting parameters from a set of demonstrations. The training set was collected with the aid of a teacher specifying good values for some different hitting locations. The learned models then were used to infer hitting parameters for unseen hitting locations. We validated the presented approach on the Barrett WAM arm in playing minigolf on two advanced fields. A summary of this study is provided in Appendix D.

Another research attempt was also carried out in our laboratory as a student semester project, conducted under my co-supervision, to play minigolf when one or more obstacles were present on the field (see Appendix F for the project definition). In this situations, the robot should be able to adopt different hitting strategies based on the configuration of objects, the position of the hole and the ball on the field.

7.3 Final Words

During the last four years, I have put a considerable amount of efforts into endowing robots with an inkling of the coordination abilities that we humans take for granted. I have found the DS approach an amazing way of modeling

robot motion primitives due to the inherent robustness, reactivity, and adaptability that it offers. However, there is a key missing feature in the current implementation of DS-based approaches. I would call it *"smartness"* and define it, with gross simplification, as the combination of the abilities to autonomously 1) generate a library of motion primitives, 2) combine and/or sequence motion primitives to perform more complex tasks, and 3) switch between them in case of perturbations.

Throughout this thesis we have assumed that all demonstrations are nicely trimmed so that the first and last points in each demonstration correspond to the onset and the end of the desired motion, respectively. But, in many real-life situations, the robot may face demonstrations of a complex task that can be decomposed into a sequence of basic motions. It would be advantageous if the robot could autonomously segment these demonstrations into their basic components and use these either to learn a new motion primitive or to adapt a previously learned model.

Sequencing of motion primitives is also a key requirement in order to avoid storing a large library of motion primitives, as well as to perform complex tasks in a more efficient way. The ability to switch between motion primitives could be essential in the case of perturbations. If the working environment of the robot significantly changes during the execution of a task, it might be better to choose an alternative motion primitive rather than insisting on executing the current one.

Despite many works that have been done on segmenting and sequencing motions, only a few DS-based approaches have been developed so far on these topics, and with very limited capability. I believe further researches should be envisaged along this direction in order to bring us closer to our vision of having smart robotic systems with a high level of adaptability, reactivity, and robustness.

Appendices

Appendix A

Proofs of Theorems

A.1 Proof of Theorem 4.1

First, without loss of generality, we can assume that the target $\boldsymbol{\xi}^*$ is located at the origin. We start the proof by recalling the following from (Pettersson & Lennartson, 1997; Borne & Dieulot, 2005). Suppose there exist a continuously differentiable nonlinear system for which a piecewise Lyapunov function $V^k(\boldsymbol{\xi})$ is defined for each subdomain Ω^k, $k = 1..K$. If $\forall \boldsymbol{\xi} \in \Omega^k$ there exist positive constants α^k, β^k, and $s > 0$ such that:

$$\alpha^k \|\boldsymbol{\xi}\|^s \leq V^k(\boldsymbol{\xi}) \leq \beta^k \|\boldsymbol{\xi}\|^s \qquad \forall \boldsymbol{\xi} \in \Omega^k, k = 1..K \tag{A.1a}$$

$$\dot{V}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) < 0 \qquad \forall \boldsymbol{\xi} \in \Omega^k \setminus \mathbf{0}, k = 1..K \tag{A.1b}$$

$$V^k(\boldsymbol{\xi}) < V^{k-1}(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \boldsymbol{\Phi}^k, k = 2..K \tag{A.1c}$$

$$V^K(\mathbf{0}) = 0 \tag{A.1d}$$

$$\dot{V}^K(\mathbf{0}) = 0 \tag{A.1e}$$

then the origin ($\boldsymbol{\xi}^* = \mathbf{0}$) is asymptotically stable in the sense of Lyapunov[1]. Consequently, given a system described by Eq. (4.17), and a positive scalar b^k, for every subregion Ω^k, we define a Lyapunov function $V^k(\boldsymbol{\xi})$ of the form:

$$\begin{cases} V^1(\boldsymbol{\xi}) = \frac{1}{E^1(\boldsymbol{\xi})} + \rho^1 & \forall \boldsymbol{\xi} \in \Omega^1 \\ V^k(\boldsymbol{\xi}) = \frac{E^{k-1}(\boldsymbol{\xi})}{E^k(\boldsymbol{\xi})} + \rho^k & \forall \boldsymbol{\xi} \in \Omega^k, k \in 2..K \end{cases} \tag{A.2}$$

where $E^k(\boldsymbol{\xi}) = e^{-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu}_{\boldsymbol{\xi}}^k)}$, $\forall k \in 1..K$. Note that by construction $V^k(\boldsymbol{\xi})$ is positive, bounded, continuous and continuously differentiable in Ω^k, $\forall k = 1..K$. It can be easily shown that there always exist positive scalar α^k and β^k such that conditions Eq. (A.1a) is satisfied. Similarly, one can find a set of positive scalar ρ^k to satisfy condition Eqs. (A.1c) and (A.1d).

In order to ensure Eq. (A.1b), we start by taking the derivative of V^k: [2]

$$\dot{V}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \frac{\dot{E}^{k-1}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) E^k(\boldsymbol{\xi}) - E^{k-1}(\boldsymbol{\xi}) \dot{E}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})}{\left(E^k(\boldsymbol{\xi})\right)^2} \tag{A.3}$$

[1] Regarding the system described in Section 4.4.1, the only possible transitions are from subregions Ω^k to Ω^{k+1} via hyperplanes $\boldsymbol{\Phi}^k$ (see Eq. (4.18c)), which results in having only one transition through each $\boldsymbol{\Phi}^k$.

[2] Note that both V and E are a function of $\boldsymbol{\xi}$ while their derivatives are a function of both $\boldsymbol{\xi}$ and $\dot{\boldsymbol{\xi}}$.

$\dot{V}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})$ is ensured to be negative definite in each subregion Ω^k if:

$$
\begin{aligned}
& \dot{V}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) < 0 \\
\Leftrightarrow\ & \dot{E}^{k-1}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) E^k(\boldsymbol{\xi}) - E^{k-1}(\boldsymbol{\xi}) \dot{E}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) < 0 \\
\Leftrightarrow\ & \frac{\dot{E}^{k-1}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})}{E^{k-1}(\boldsymbol{\xi})} < \frac{\dot{E}^k(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})}{E^k(\boldsymbol{\xi})} \\
\Leftrightarrow\ & \frac{\frac{\partial E^{k-1}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \dot{\boldsymbol{\xi}}}{E^{k-1}(\boldsymbol{\xi})} < \frac{\frac{\partial E^k(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \dot{\boldsymbol{\xi}}}{E^k(\boldsymbol{\xi})} \\
\Leftrightarrow\ & \frac{-(\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k-1})^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{k-1})^{-1} E^{k-1}(\boldsymbol{\xi}) \dot{\boldsymbol{\xi}}}{E^{k-1}(\boldsymbol{\xi})} < \frac{-(\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k})^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{k})^{-1} E^{k}(\boldsymbol{\xi}) \dot{\boldsymbol{\xi}}}{E^{k}(\boldsymbol{\xi})} \\
\Leftrightarrow\ & (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k-1})^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{k-1})^{-1} \dot{\boldsymbol{\xi}} > (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{k})^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{k})^{-1} \dot{\boldsymbol{\xi}}
\end{aligned} \tag{A.4}
$$

Similarly, in Ω^1 we have:

$$
\dot{V}^1(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) < 0 \quad \Leftrightarrow \quad (\boldsymbol{\xi} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^{1})^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^{1})^{-1} \dot{\boldsymbol{\xi}} < 0 \tag{A.5}
$$

The conditions given by Eqs. (A.4) and (A.5) are satisfied as they are imposed as hard constraints when building an estimate of the DS (see Eq. (4.18b)). Hence, the condition given by Eq. (A.1b) is satisfied over the region D.

Finally, following the derivation carried out above, the condition given by Eq. (A.1e) holds if the velocity $\dot{\boldsymbol{\xi}}$ vanishes at the target $\boldsymbol{\xi}^* = \mathbf{0} \in \Omega^K$. Solving Eq. (4.17) for $\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\mathbf{0}) = \mathbf{0}$ yields:

$$
h^{K-1}(\mathbf{0}) \boldsymbol{b}^{K-1} + h^K(\mathbf{0}) \boldsymbol{b}^K = \mathbf{0}
$$

Substituting $\boldsymbol{b}^k$ with its equivalence from Eq. (4.9) and using the stability condition given by Eq. (4.18a), we obtain $\boldsymbol{f}(\mathbf{0}) = \mathbf{0}$, and by extension $\dot{V}^K(\mathbf{0}) = 0$. Note that without having this condition, it is impossible to find appropriate α^k, β^k, and $s > 0$ to bound the Lyapunov function around the origin.

Thus the conditions given by Eq. (A.1) are satisfied over the region D, and thus the DS $\boldsymbol{f}(\boldsymbol{\xi})$ is locally asymptotically stable at the target $\boldsymbol{\xi}^*$.

A.2 Proof of Theorem 4.2

We start the proof by recalling the Lyapunov conditions for asymptotic stability of an arbitrary autonomous DS (Slotine & Li, 1991):

Lyapunov Stability Theorem: *A dynamical system determined by the function $\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi})$ is globally asymptotically stable at the point $\boldsymbol{\xi}^*$ if there exists a continuous and continuously differentiable Lyapunov function $V(\boldsymbol{\xi}) : \mathbb{R}^d \to \mathbb{R}$ such that:*

$$
V(\boldsymbol{\xi}) > 0 \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^* \tag{A.6a}
$$

$$
\dot{V}(\boldsymbol{\xi}) < 0 \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^* \tag{A.6b}
$$

$$
V(\boldsymbol{\xi}^*) = 0 \tag{A.6c}
$$

$$
\dot{V}(\boldsymbol{\xi}^*) = 0 \tag{A.6d}
$$

Consider a Lyapunov function $V(\boldsymbol{\xi})$ of the form:

$$V(\boldsymbol{\xi}) = \frac{1}{2}(\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T(\boldsymbol{\xi} - \boldsymbol{\xi}^*) \qquad \forall \boldsymbol{\xi} \in \mathbb{R}^d \tag{A.7}$$

First observe that $V(\boldsymbol{\xi})$ is a quadratic function and hence satisfies the condition given by Eq. (A.6a). Considering Eqs. (4.9) and (4.23), the condition given by Eq. (A.6b) follows from taking the first derivative of $V(\boldsymbol{\xi})$ with respect to time, we have:[3]

$$\begin{aligned}
\dot{V}(\boldsymbol{\xi}) &= \frac{dV}{dt} = \frac{dV}{d\boldsymbol{\xi}}\frac{d\boldsymbol{\xi}}{dt} \\
&= \frac{1}{2}\frac{d}{d\boldsymbol{\xi}}\left((\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T(\boldsymbol{\xi} - \boldsymbol{\xi}^*)\right)\dot{\boldsymbol{\xi}} \\
&= (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T\dot{\boldsymbol{\xi}} = (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T\boldsymbol{f}(\boldsymbol{\xi}) \\
&= (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T \underbrace{\sum_{k=1}^{K} h^k(\boldsymbol{\xi})(\boldsymbol{A}^k\boldsymbol{\xi} + \boldsymbol{b}^k)}_{=\dot{\boldsymbol{\xi}} \text{ (see Eq. (4.10))}} \\
&= (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T \sum_{k=1}^{K} h^k(\boldsymbol{\xi})\big(\boldsymbol{A}^k(\boldsymbol{\xi} - \boldsymbol{\xi}^*) + \underbrace{\boldsymbol{A}^k\boldsymbol{\xi}^* + \boldsymbol{b}^k}_{=0 \text{ (see Eq. (4.23a))}}\big) \\
&= (\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T \sum_{k=1}^{K} h^k(\boldsymbol{\xi})\boldsymbol{A}^k(\boldsymbol{\xi} - \boldsymbol{\xi}^*) \\
&= \sum_{k=1}^{K} \underbrace{h^k(\boldsymbol{\xi})}_{h^k>0}\underbrace{(\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T\boldsymbol{A}^k(\boldsymbol{\xi} - \boldsymbol{\xi}^*)}_{<0 \text{ (see Eq. (4.23b))}} \\
&< 0 \quad \forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^*
\end{aligned} \tag{A.8}$$

Conditions given by Eqs. (A.6c) and (A.6d) is satisfied when substituting $\boldsymbol{\xi} = \boldsymbol{\xi}^*$ into Eqs. (A.7) and (A.8):

$$V(\boldsymbol{\xi}^*) = \frac{1}{2}(\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T(\boldsymbol{\xi} - \boldsymbol{\xi}^*)\Big|_{\boldsymbol{\xi}=\boldsymbol{\xi}^*} = 0 \tag{A.9}$$

$$\dot{V}(\boldsymbol{\xi}^*) = \sum_{k=1}^{K} h^k(\boldsymbol{\xi})(\boldsymbol{\xi} - \boldsymbol{\xi}^*)^T\boldsymbol{A}^k(\boldsymbol{\xi} - \boldsymbol{\xi}^*)\Big|_{\boldsymbol{\xi}=\boldsymbol{\xi}^*} = 0 \tag{A.10}$$

Therefore, an arbitrary ODE function $\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi})$ given by Eq. (4.10) is globally asymptotically stable if conditions of Eq. (4.23) are satisfied.

A.3 Proof of Theorem 4.3

Following the Lyapunov stability theorem introduced in Section 2.1, the DS $\dot{\boldsymbol{\xi}} = \tilde{\boldsymbol{f}}(.)$ that is given by Eq. (4.32) and its evolution in time is computed according to Eq. (4.3) is globally asymptotically stable if we could verify $\dot{V}(\boldsymbol{\xi}; \boldsymbol{\theta}) < 0$,

[3] Note that $\dot{V}$ is a function of both $\boldsymbol{\xi}$ and $\dot{\boldsymbol{\xi}}$. However, since $\dot{\boldsymbol{\xi}}$ can be directly expressed in terms of $\boldsymbol{\xi}$ using Eq. (4.10), one can finally infer that $\dot{V}$ only depends on $\boldsymbol{\xi}$.

$\forall \boldsymbol{\xi} \in \mathbb{R}^d \backslash \boldsymbol{\xi}^*$, and $\dot{V}(\boldsymbol{\xi}^*; \boldsymbol{\theta}) = 0$. By applying the chain rule and using Eqs. (4.32), (4.34) and (4.42) to (4.46) we have:

$$\begin{aligned}
\dot{V}(.) &= \nabla_{\boldsymbol{\xi}} V(.)^T \dot{\boldsymbol{\xi}} = \nabla_{\boldsymbol{\xi}} V(.)^T \tilde{\boldsymbol{f}}(.) = \nabla_{\boldsymbol{\xi}} V(.)^T \big(\boldsymbol{f}(.) + \boldsymbol{u}(.)\big) \\
&= \nabla_{\boldsymbol{\xi}} V(.)^T \boldsymbol{f}(.) - \phi(.)\Big(\alpha(.) + \rho(.)\Big) \nabla_{\boldsymbol{\xi}} V(.)^T \bar{\boldsymbol{v}}(.) \\
&= \|\nabla_{\boldsymbol{\xi}} V(.)\| \frac{\nabla_{\boldsymbol{\xi}} V(.)^T \boldsymbol{f}(.)}{\|\nabla_{\boldsymbol{\xi}} V(.)\|} - \phi(.)\Big(\alpha(.) + \rho(.)\Big) \frac{\nabla_{\boldsymbol{\xi}} V(.)^T \nabla_{\boldsymbol{\xi}} V(.)}{\|\nabla_{\boldsymbol{\xi}} V(.)\|} \\
&= \alpha(.)\|\nabla_{\boldsymbol{\xi}} V(.)\| - \phi(.)\Big(\alpha(.) + \rho(.)\Big)\|\nabla_{\boldsymbol{\xi}} V(.)\| \\
&= \alpha(.)\big(1 - \phi(.)\big)\|\nabla_{\boldsymbol{\xi}} V(.)\| - \phi(.)\rho(.)\|\nabla_{\boldsymbol{\xi}} V(.)\| \\
&= \Big(\alpha(.)\big(1 - \phi(.)\big) - \phi(.)\rho(.)\Big)\|\nabla_{\boldsymbol{\xi}} V(.)\| \qquad \text{(A.11)}
\end{aligned}$$

To verify the negativity of $\dot{V}(.)$ in $\mathbb{R}^d \backslash \boldsymbol{\xi}^*$, we consider three cases based on the value of $\alpha(.)$:

1. For $\alpha(.) < -\pi/\tau$: Considering Eq. (4.45), $\phi(.) = 0$ and thus the first and second terms are always negative and zero, respectively.

2. For $-\pi/\tau \leq \alpha(.) \leq 0$: First observe that $\rho(.) > 0$ and that $0 \leq \phi(.) \leq 1$. In this case each term in Eq. (A.11) is less than or equal to zero, and the net effect of both terms are always less than zero.

3. For $\alpha(.) > 0$: For positive values of α we have $\phi(.) = 1$. Therefore the first term is always zero and the second term is less than zero which verifies $\dot{V}(.) < 0$.

Thus the rate of change in energy function is always negative $\forall \boldsymbol{\xi} \in \mathbb{R}^d \backslash \boldsymbol{\xi}^*$. At the target point, by construction we have $\nabla_{\boldsymbol{\xi}} V(\boldsymbol{\xi}^*; \boldsymbol{\theta}) = 0$ which verifies $\dot{V}(\boldsymbol{\xi}^*; \boldsymbol{\theta}) = 0$ (see Eq. (4.38)).

Note that for non-autonomous DS, further evaluation should be done in order to ensure global uniform asymptotic stability at $\boldsymbol{\xi}^*$. First observe that in our formulation both the energy function and the target point are time-invariant, but the time derivative of the energy function is time-dependent since $\frac{d}{dt} V(\boldsymbol{\xi}; \boldsymbol{\theta}) = \nabla V(\boldsymbol{\xi}; \boldsymbol{\theta})^T \tilde{\boldsymbol{f}}(t, \boldsymbol{\xi}) = \dot{V}(t, \boldsymbol{\xi}; \boldsymbol{\theta})$. Let us now consider an arbitrary initial point $\boldsymbol{\xi}(0)$ with its associated energy function $V(\boldsymbol{\xi}(0); \boldsymbol{\theta})$. If $\boldsymbol{\xi}(0) = \boldsymbol{\xi}^*$, then $\boldsymbol{\xi}(t) = \boldsymbol{\xi}^*$ for all t because otherwise $V(\boldsymbol{\xi}; \boldsymbol{\theta})$ would have to increase which contradicts the previous proof. If $\boldsymbol{\xi}(0) \neq \boldsymbol{\xi}^*$, then $V(\boldsymbol{\xi}(t); \boldsymbol{\theta})$ decreases strictly, hence the whole trajectory lies in the bounded level set defined by $V(\boldsymbol{\xi}; \boldsymbol{\theta}) = V(\boldsymbol{\xi}(0); \boldsymbol{\theta})$. Now, let us define the subset of points S_ε for some $\varepsilon > 0$ by:

$$S_\varepsilon = \{\boldsymbol{\xi} \in \mathbb{R}^d \mid \varepsilon \leq V(\boldsymbol{\xi}; \boldsymbol{\theta}) \leq V(\boldsymbol{\xi}(0); \boldsymbol{\theta})\} \qquad \text{(A.12)}$$

S_ε is a bounded and closed set. Therefore any continuous function on S_ε takes its maximum within S_ε. If we define:

$$m_\varepsilon = \max_{\boldsymbol{\xi} \in S_\varepsilon,\ t \in [0\ \infty)} \dot{V}(t, \boldsymbol{\xi}(t); \boldsymbol{\theta}) \qquad \text{(A.13)}$$

then $m_\varepsilon < 0$ because $\boldsymbol{\xi}^* \notin S_\varepsilon$ and $\dot{V}(t, \boldsymbol{\xi}(t); \boldsymbol{\theta}) < 0$, $\forall \boldsymbol{\xi} \in \mathbb{R}^d \setminus \boldsymbol{\xi}^*$. Thus, we have:

$$\dot{V}(t, \boldsymbol{\xi}(t); \boldsymbol{\theta}) \leq m_\varepsilon < 0 \quad \Rightarrow \quad V(\boldsymbol{\xi}(t); \boldsymbol{\theta}) \leq V(\boldsymbol{\xi}(0); \boldsymbol{\theta}) + m_\varepsilon t \tag{A.14}$$

as long as $\boldsymbol{\xi}(t) \in S_\varepsilon$. However since $V(\boldsymbol{\xi}(t); \boldsymbol{\theta}) > 0$, this can only be true for a finite interval of time. Consequently, there is a time $t_\varepsilon < \infty$ such that $V(\boldsymbol{\xi}(t); \boldsymbol{\theta}) < \varepsilon$ for some $t > t_\varepsilon$. Since this is true for any $\varepsilon > 0$, we have $\lim_{t \to \infty} V(\boldsymbol{\xi}(t); \boldsymbol{\theta}) = 0$, and thus by extension $\lim_{t \to \infty} \boldsymbol{\xi}(t) = \boldsymbol{\xi}^*$. Furthermore, as the above conclusion is true $\forall \boldsymbol{\xi}(0) \in \mathbb{R}^d$, the system is globally uniformly asymptotically stable at the target.

A.4 Proof of Theorem 6.1

Consider a hyper-surface $\boldsymbol{\mathcal{X}}^b \subset \mathbb{R}^d$ corresponding to boundary points of a hyper-sphere obstacle in $\mathbb{R}^d$ with a center $\boldsymbol{\xi}^o$ and a radius r^o. Impenetrability of the obstacle's boundaries is ensured if the normal velocity at boundary points $\boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b$ vanishes:

$$\boldsymbol{n}(\boldsymbol{\xi}^b)^T \, \dot{\boldsymbol{\xi}}^b = 0 \qquad\qquad \forall \boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b \tag{A.15}$$

where $\boldsymbol{n}(\boldsymbol{\xi}^b)$ is the unit normal vector at a boundary point $\boldsymbol{\xi}^b$:

$$\boldsymbol{n}(\boldsymbol{\xi}^b) = \frac{\boldsymbol{\xi}^b - \boldsymbol{\xi}^o}{\|\boldsymbol{\xi}^b - \boldsymbol{\xi}^o\|} \xrightarrow{\tilde{\boldsymbol{\xi}}^b = \boldsymbol{\xi}^b - \boldsymbol{\xi}^o} \boldsymbol{n}(\boldsymbol{\xi}^b) = \frac{\tilde{\boldsymbol{\xi}}^b}{r^o} \qquad \forall \boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b \tag{A.16}$$

The eigenvalue decomposition of the square matrix $\boldsymbol{M}^s(\tilde{\boldsymbol{\xi}}, r^o)$ is given by:

$$\boldsymbol{M}^s(\tilde{\boldsymbol{\xi}}, r^o) = \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}, r^o) \, \boldsymbol{D}^s(\tilde{\boldsymbol{\xi}}, r^o) \, \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}, r^o)^{(-1)} \tag{A.17}$$

where $\boldsymbol{D}^s(\tilde{\boldsymbol{\xi}}, r^o)$ is a $d \times d$ diagonal matrix composed of the eigenvalues:

$$\begin{cases} \lambda^1 = 1 - \frac{(r^o)^2}{\tilde{\boldsymbol{\xi}}^T \tilde{\boldsymbol{\xi}}} \\ \lambda^i = 1 + \frac{(r^o)^2}{\tilde{\boldsymbol{\xi}}^T \tilde{\boldsymbol{\xi}}} \forall i \in 2..d \end{cases} \tag{A.18}$$

and $\boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}, r^o) = [\boldsymbol{v}^1 \quad \cdots \quad \boldsymbol{v}^d]$ is the matrix of eigenvectors with:

$$\begin{cases} \boldsymbol{v}^1 = \tilde{\boldsymbol{\xi}} \\ \\ \boldsymbol{v}^i_j = \begin{cases} -\tilde{\boldsymbol{\xi}}_i & j = 1 \\ \tilde{\boldsymbol{\xi}}_1 & j = i \\ 0 & j \neq 1, i \end{cases} \qquad \forall i \in 2..d, \; j \in 1..d \end{cases} \tag{A.19}$$

Substituting Eqs. (6.4), (A.16) and (A.17) into Eq. (A.15) yields:

$$\boldsymbol{n}(\boldsymbol{\xi}^b)^T \, \dot{\boldsymbol{\xi}}^b = \frac{(\tilde{\boldsymbol{\xi}}^b)^T}{r} \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{D}^s(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{f}(.) \tag{A.20}$$

Since $\boldsymbol{\xi}^b$ is equal to the first eigenvector of $\boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}^b, r^o)$, Eq. (A.20) reduces to:

$$\boldsymbol{n}(\boldsymbol{\xi}^b)^T \, \dot{\boldsymbol{\xi}}^b = \begin{bmatrix} r^o \\ [\boldsymbol{0}]_{d-1} \end{bmatrix}^T \boldsymbol{D}^s(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{f}(.) \tag{A.21}$$

where $[\boldsymbol{0}]_{d-1}$ is a zero column vector of dimension $d-1$. For all points on the obstacle boundary, the first eigenvalue is zero, i.e. $\lambda^1 = 0$, $\forall \boldsymbol{\xi}^b \in \boldsymbol{\mathcal{X}}^b$. Thus, we have:

$$\boldsymbol{n}(\boldsymbol{\xi}^b)^T \, \dot{\boldsymbol{\xi}}^b = [\boldsymbol{0}]_d^T \, \boldsymbol{V}^s(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{f}(.) = 0 \tag{A.22}$$

A.5 Proof of Theorem 6.2

The proof of Theorem 6.2 follows directly from that of Theorem 6.1:

$$\boldsymbol{n}(\boldsymbol{\xi}^b)^T \boldsymbol{R}^T \dot{\boldsymbol{\xi}}^b = \boldsymbol{n}(\boldsymbol{\xi}^b)^T \underbrace{\boldsymbol{R}^T \boldsymbol{R}}_{\boldsymbol{I}} \boldsymbol{E}(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{D}(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{E}(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{R}^T \boldsymbol{f}(.) \tag{A.23}$$

Considering the fact that $\boldsymbol{n}(\boldsymbol{\xi}^b)$ is equal to the first eigenvector of $\boldsymbol{E}(\tilde{\boldsymbol{\xi}}^b, r^o)$, and the first eigenvalue is zero for all points on the obstacle boundary yields:

$$\begin{aligned} \boldsymbol{n}(\boldsymbol{\xi}^b)^T \, \dot{\boldsymbol{\xi}}^b &= \begin{bmatrix} 1 \\ [\boldsymbol{0}]_{d-1} \end{bmatrix}^T \boldsymbol{D}(\tilde{\boldsymbol{\xi}}^b, r^o) \, \boldsymbol{E}(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{f}(.) \boldsymbol{R}^T \\ &= [\boldsymbol{0}]_d^T \, \boldsymbol{E}(\tilde{\boldsymbol{\xi}}^b, r^o)^{(-1)} \boldsymbol{R}^T \boldsymbol{f}(.) = 0 \end{aligned} \tag{A.24}$$

Appendix B

Qualitative Comparison across BM, SEDS, and SEDS-II on the Library of 2D Handwriting Motions

This appendix provides supplementary results to Section 4.7 on the estimate of 20 human handwriting motions. The comparison was made between BM, two variants of SEDS, and four variants of SEDS-II. The demonstrations are collected from pen input using a Tablet-PC. For each motion, the evaluation is made on a set of six test trajectories that are spread in between and outside the training trajectories. The training and test trajectories are shown in Fig. 4.35. The qualitative comparison across these approaches are provided in Figs. B.1 to B.7. For information about the quantitative comparison between these methods, please refer to Section 4.7.

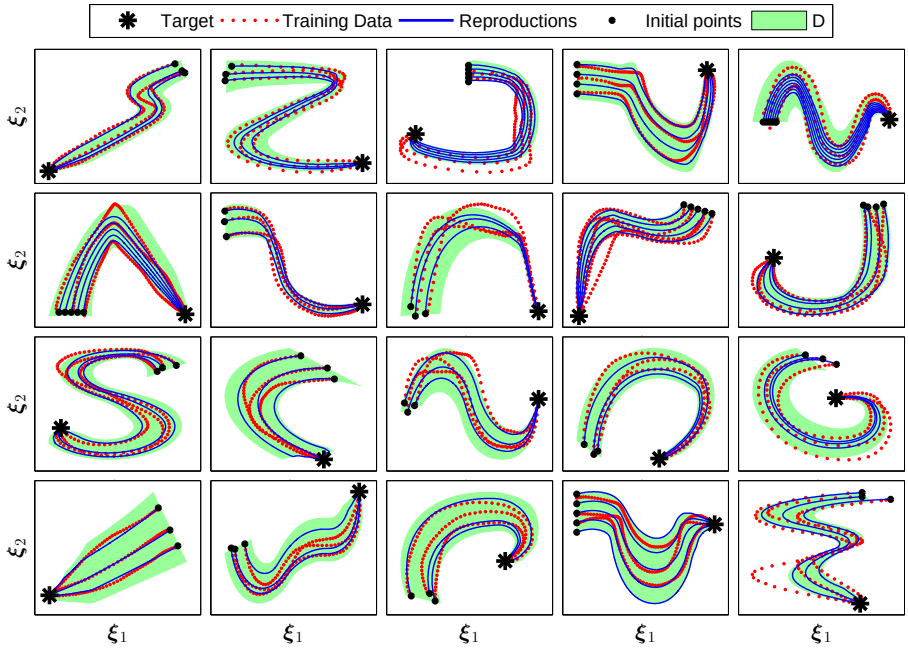


Figure B.1: Qualitative performance evaluation of BM in learning 20 nonlinear 2D motions.

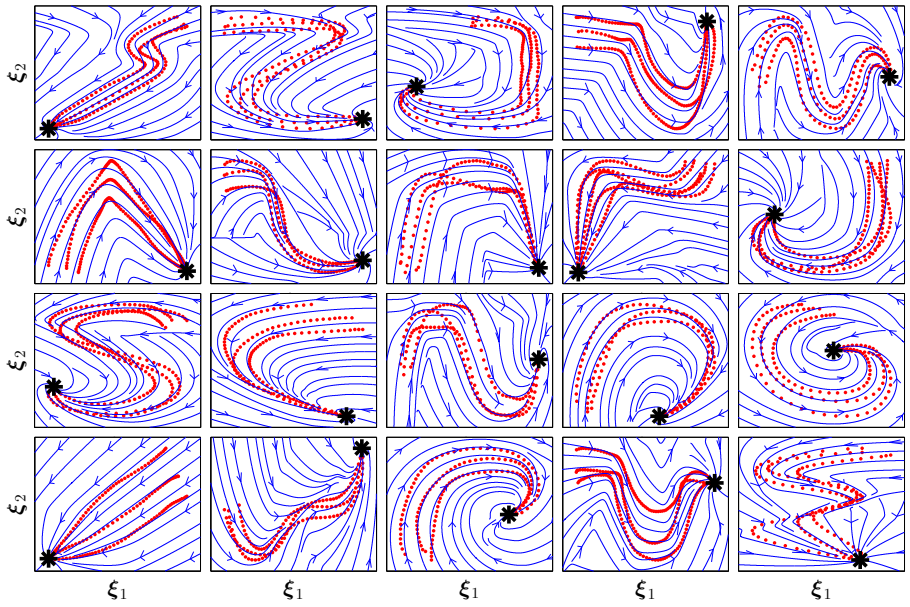


Figure B.2: Qualitative performance evaluation of SEDS-Likelihood in learning 20 nonlinear 2D motions.

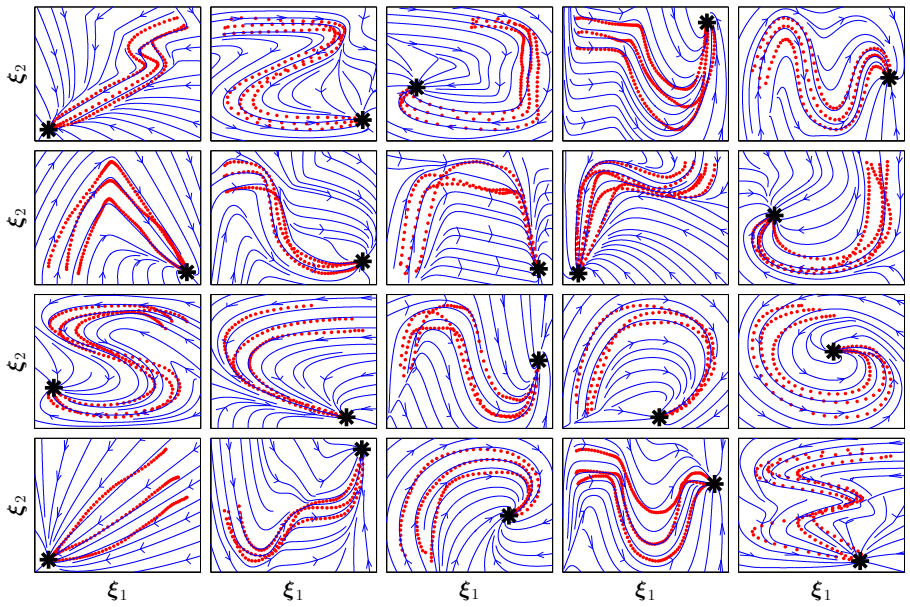


Figure B.3: Qualitative performance evaluation of SEDS-MSE in learning 20 nonlinear 2D motions.

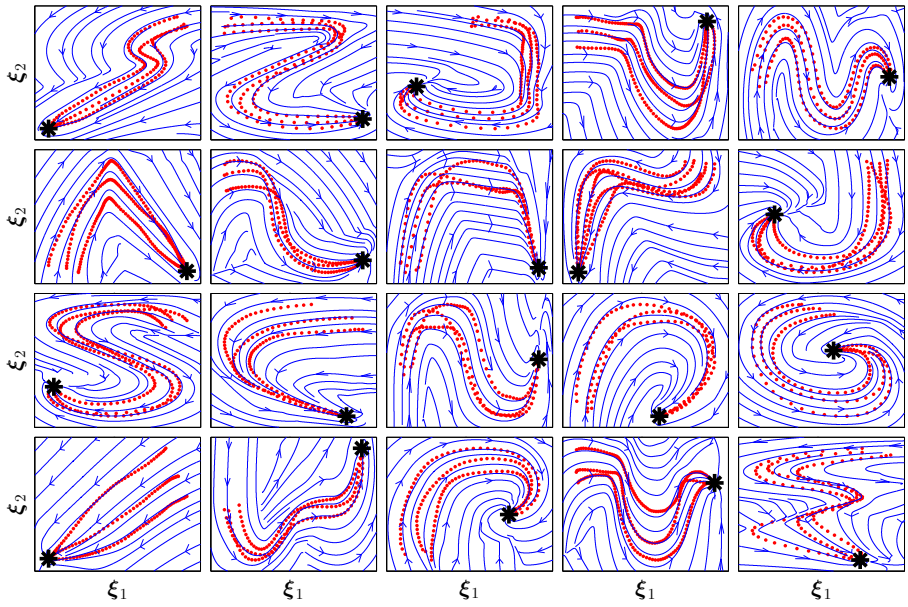


Figure B.4: Qualitative performance evaluation of SEDS-II with GMR encoding in learning 20 nonlinear 2D motions.

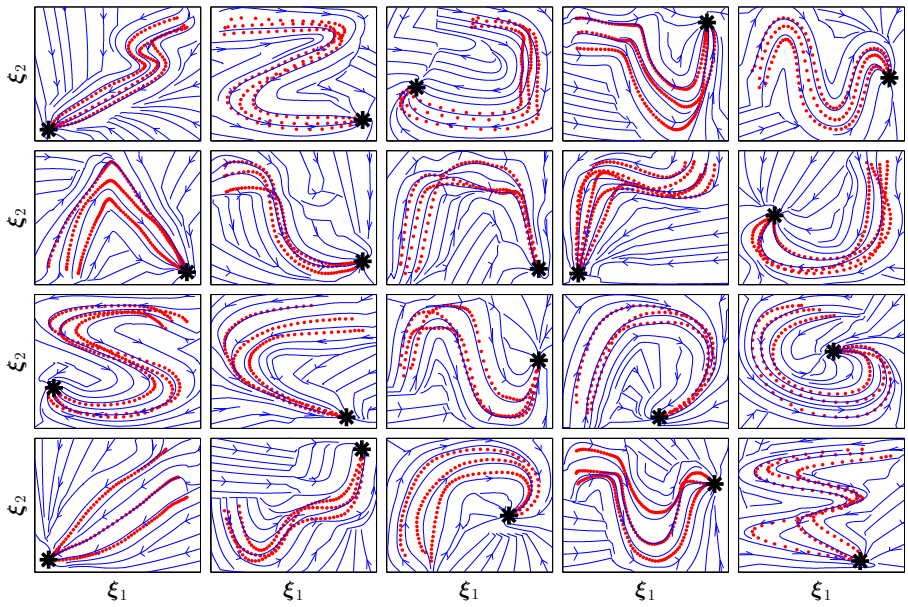


Figure B.5: Qualitative performance evaluation of SEDS-II with LWPR encoding in learning 20 nonlinear 2D motions.

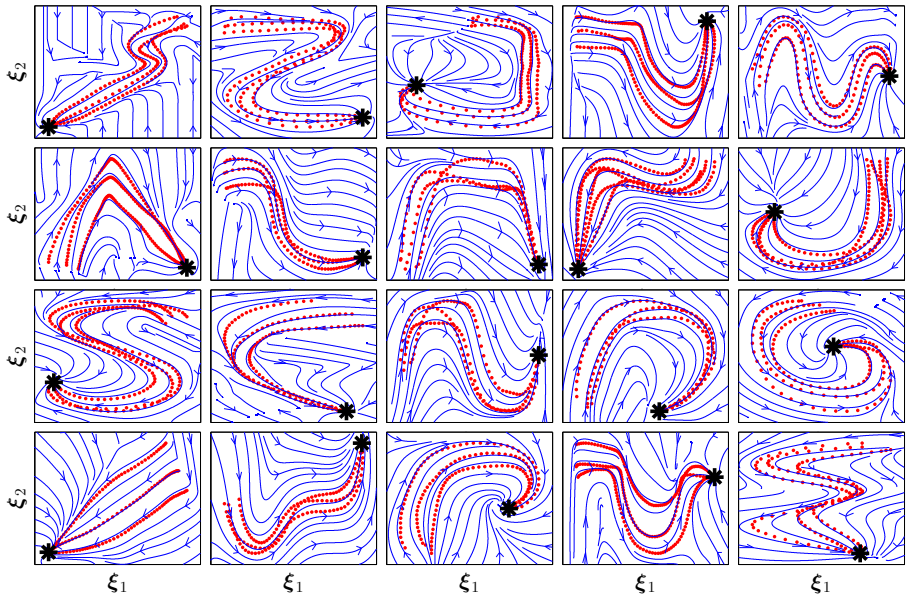


Figure B.6: Qualitative performance evaluation of SEDS-II with GPR encoding in learning 20 nonlinear 2D motions.

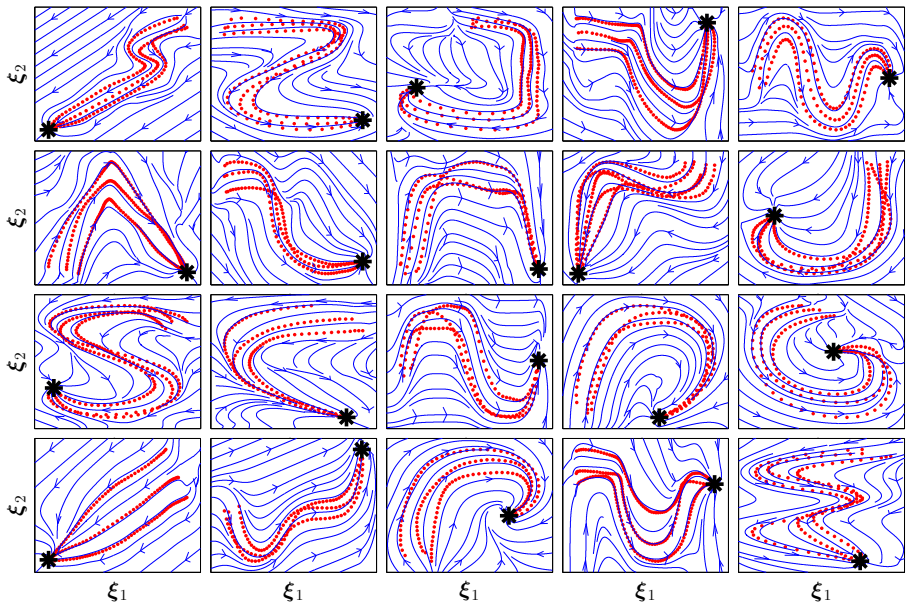


Figure B.7: Qualitative performance evaluation of SEDS-II with SVR encoding in learning 20 nonlinear 2D motions.

Appendix C

Analytical Computation of Derivatives for SEDS

This technical report provides supplementary information for the SEDS optimization problems defined in Section 4.5. Reading of this appendix is *not necessary* for researchers who only want to use SEDS learning algorithm. This appendix is aimed at helping those who want to develop SEDS, or to write their own optimization program. All the formulations reported here are developed for SEDS models; however, they can also be used for general GMM formulations. In the case of the latter, they should be slightly modified to consider the general form of GMM. Hopefully, this appendix would be clear enough to help readers for doing that.

To facilitate reading of this section, a list of main variables and mathematical notations is provided in Table C.1. Furthermore, to have a clean summary of the final results, all the derivatives are summarized in Tables C.2 to C.6.

The remainder of this document is structured as follows. Appendices C.1 and C.3 provide analytical formulations to compute the derivatives of MSE and Likelihood cost functions with respect to the optimization parameters, respectively. In addition, Appendices C.2 and C.4 present two alternative optimization problems that automatically satisfy 4 out of 5 constraints of the original optimization problem through a change of variable. Finally, Appendix C.5 provides the analytical derivatives of the optimization's constraints with respect to the optimization parameters.

Table C.1: Nomenclature

Variable	Type (size)	Description
d	Scalar	Dimension of DS
$\boldsymbol{\xi}$	Vector (d)	Input variable, e.g. position
$\boldsymbol{\xi}^*$	Vector (d)	Target point
$\dot{\boldsymbol{\xi}}$	Vector (d)	Output variable, e.g. velocity
π	Scalar	Prior of the Gaussian function
$\boldsymbol{\mu}$	Vector ($2d$)	Center of the Gaussian function
$\boldsymbol{\Sigma}$	Matrix ($2d \times 2d$)	Covariance matrix of the Gaussian function
$\boldsymbol{f}$	Function ($d \mapsto d$)	Unknown original DS
J	Scalar	Optimization cost function
$\boldsymbol{\theta}$	Structure	Optimization parameters
$\boldsymbol{L}$	Matrix ($2d \times 2d$)	Lower triangle matrix
$\boldsymbol{A}$	Matrix ($d \times d$)	Matrix of the linear DS
$\boldsymbol{b}$	Vector (d)	Intersection point of the linear DS
$\boldsymbol{I}$	Matrix	Identity matrix
$\mathbf{0}$	Vector	Zero vector
K	Scalar	Number of Gaussian functions
N	Scalar	Number of demonstrations
$\mathcal{T}$	Scalar	Total number of training data points

Notation	Description
$(.)^k$	Of the k-th Gaussian function
$(.)^T$	Transpose of a Vector/matrix
$(.)^{t,n}$	The t-th datapoint of the n-th demonstration
$(.)_i$	The i-th component of a vector
$(.)_{ij}$	The (i,j)-th component of a matrix
$(\text{vec})_{\boldsymbol{\xi}}$	Sub-vector of vec with indices $1:d$
$(\text{vec})_{\dot{\boldsymbol{\xi}}}$	Sub-vector of vec with indices $d{+}1:2d$
$(\text{mat})_{\boldsymbol{\xi}}$	Sub-matrix of mat with indices $(1:d, 1:d)$
$(\text{mat})_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}$	Sub-matrix of mat with indices $(d{+}1:2d, 1:d)$
$(.)_{1:c,1:c}$	A slice of a matrix with indices $(1:c, 1:c)$
$\mathbf{0}^{\{i\}}$	A zero vector with the exception that its i-th component is 1
$\mathbf{0}^{\{ij\}}$	A matrix of zeros with the exception that its (i,j)-th comp. is 1
$\mathbf{0}^{\{\overline{ij}\}}$	A matrix of zeros with the exception that its (i,j) and (j,i)-th components are one.
$adj(.)$	Adjugate of a matrix
$tr(.)$	Trace of a matrix
$ln(.)$	The natural logarithm
$Chol(.)$	Cholesky decomposition of a matrix

Table C.2: Derivatives of the MSE cost function (taken from Appendix C.1).

$\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^1..\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^K; \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^1..\boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^K\}$

Cost function: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$

Indices range: $k \in 1..K, \quad i \in 1..d$

$\frac{\partial J}{\partial \pi^k} = \frac{1}{\pi^k \mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n})(\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$

$\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} = \frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n}) \left((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \mathbf{0}^{\{i\}} \right) (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$

$\frac{\partial J}{\partial \boldsymbol{\Sigma}_{\boldsymbol{\xi},ij}^k} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n})(\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T \Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)(\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) - tr\big((\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}}\big)(\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) - 2\boldsymbol{A}^k \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\xi}^{t,n} \Big) \qquad j \in 1..i$

$\frac{\partial J}{\partial \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi},ij}^k} = \frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n})(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T \mathbf{0}^{\{ij\}} (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\xi}^{t,n} \qquad j \in 1..d$

C.1 Mean Square Error Optimization

Mean Square Error (MSE) is a means to quantify the accuracy of estimations based on demonstrations, and it is defined as:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\dot{\boldsymbol{\xi}}^{t,n} - -\dot{\boldsymbol{\xi}}^{t,n}) \tag{C.1}$$

subject to

$$b^k = -\boldsymbol{A}^k \boldsymbol{\xi}^* \tag{C.2a}$$

$$\boldsymbol{A}^k + (\boldsymbol{A}^k)^T \prec 0 \tag{C.2b}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k \succ 0 \tag{C.2c}$$

$$0 < \pi^k \leq 1 \tag{C.2d}$$

$$\sum_{k=1}^{K} \pi^k = 1 \tag{C.2e}$$

where $\dot{\boldsymbol{\xi}}^{t,n} = \boldsymbol{f}(\boldsymbol{\xi}^{t,n})$ are computed from Eq. (4.10), and $\mathcal{T} = \sum_{n=1}^{N} T^n$ is the total number of training data points. Note that Eq. (C.2) is obtained by substituting Eq. (4.9) into Eq. (4.25). The optimization parameters for this objective function are: $\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^1..\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^K; \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^1..\boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^K\}$. Solving the above optimization requires a user to provide the derivative of the cost function w.r.t. the optimization parameters. These derivatives are provided next.

Table C.3: Derivatives of the alternative MSE cost function (taken from Appendix C.2).

$\boldsymbol{\theta} = \{\tilde{\pi}^1..\tilde{\pi}^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{L}_{\boldsymbol{\xi}}^1..\boldsymbol{L}_{\boldsymbol{\xi}}^K; \boldsymbol{A}^1..\boldsymbol{A}^K\}$

Cost function: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$

Indices range: $k \in 1..K, \quad i \in 1..d$

Change of variables: $\tilde{\pi}^k = ln(\pi^k), \quad \boldsymbol{L}_{\boldsymbol{\xi}}^k = Chol(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)$

$$\frac{\partial J}{\partial \tilde{\pi}^k} = \frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n})(\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$$

$$\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} = \frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n}) \left((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \mathbf{0}^{\{i\}} \right) (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$$

$$\frac{\partial J}{\partial \boldsymbol{L}_{\boldsymbol{\xi},ij}^k} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n}) \Bigg((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\Phi} (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k) - \cdots \\ tr\Big((\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\Phi} \Big) \Bigg) (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})$$

where $\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}} (\boldsymbol{L}_{\boldsymbol{\xi}}^k)^T + \boldsymbol{L}_{\boldsymbol{\xi}}^k (\mathbf{0}^{\{ij\}})^T \qquad j \in 1..i$

$$\frac{\partial J}{\partial \boldsymbol{A}_{ij}^k} = \frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n})(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T \mathbf{0}^{\{ij\}} \boldsymbol{\xi}^{t,n} \qquad j \in 1..d$$

Reconstruction of GMM from the optimization parameters:

$$\pi^k = e^{\tilde{\pi}^k} / (\textstyle\sum_{i=1}^{K} e^{\tilde{\pi}^i}), \quad \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k = \boldsymbol{L}_{\boldsymbol{\xi}}^k (\boldsymbol{L}_{\boldsymbol{\xi}}^k)^T, \quad \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^k = \boldsymbol{A}^k \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k$$

Table C.4: Derivatives of the Likelihood cost function taken from Appendix C.3.

$\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{\Sigma}^1..\boldsymbol{\Sigma}^K\}$

Cost function: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \log \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | \boldsymbol{\theta})$

Indices range: $k \in 1..K$

$$\frac{\partial J}{\partial \pi^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \left(\frac{\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | k)}{\mathcal{P}^{t,n}} - 1 \right)$$

$$\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | k)}{\mathcal{P}^{t,n}} (\mathbf{0}^{\{i\}})^T \begin{bmatrix} \boldsymbol{I} & (\boldsymbol{A}^k)^T \end{bmatrix} (\boldsymbol{\Sigma}^k)^{-1} \left([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] - \boldsymbol{\mu}^k \right)$$

$$\frac{\partial J}{\partial \boldsymbol{\Sigma}_{ij}^k} = -\frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | k)}{\mathcal{P}^{t,n}} \Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k) - \\ tr\big((\boldsymbol{\Sigma}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}} \big) + 2 (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \boldsymbol{\mathcal{S}}^k \Big) \qquad \forall i \in 1..2d, \quad j \in 1..i$$

where $\boldsymbol{\mathcal{S}}^k = \begin{bmatrix} \mathbf{0} \\ \Big(-\boldsymbol{A}^k \big[\mathbf{0}^{\{\overline{ij}\}}\big]_{\boldsymbol{\xi}} + \big[\mathbf{0}^{\{\overline{ij}\}}\big]_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} \Big) (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\mu}_{\boldsymbol{\xi}}^k \end{bmatrix}$

Table C.5: Derivatives of the Likelihood cost function taken from Appendix C.4.

$\boldsymbol{\theta} = \{\tilde{\pi}^1..\tilde{\pi}^K; \boldsymbol{\mu}^1_{\boldsymbol{\xi}}..\boldsymbol{\mu}^K_{\boldsymbol{\xi}}; \boldsymbol{L}^1..\boldsymbol{L}^K\}$

Cost function: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \log \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|\boldsymbol{\theta})$

Indices range: $k \in 1..K$

Change of variables: $\tilde{\pi}^k = ln(\pi^k), \quad \boldsymbol{L}^k = Chol(\boldsymbol{\Sigma}^k)$

$\frac{\partial J}{\partial \tilde{\pi}^k} = -\frac{e^{\tilde{\pi}^k}}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \left(\frac{\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} - 1 \right)$

$\frac{\partial J}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} (\mathbf{0}^{\{i\}})^T \left[\boldsymbol{I} \quad (\boldsymbol{A}^k)^T \right] (\boldsymbol{\Sigma}^k)^{-1} \left([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] - \boldsymbol{\mu}^k \right)$

$\frac{\partial J}{\partial \boldsymbol{L}^k_{ij}} = -\frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} \Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \boldsymbol{\Phi} (\boldsymbol{\Sigma}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k) \cdots$

$$-tr\big((\boldsymbol{\Sigma}^k)^{-1}\boldsymbol{\Phi}\big) + 2(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \tilde{\boldsymbol{\mathcal{S}}}^k \Big) \qquad i \in 1..2d, \quad j \in 1..i$$

where $\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}} (\boldsymbol{L}^k)^T + \boldsymbol{L}^k (\mathbf{0}^{\{ij\}})^T, \quad \tilde{\boldsymbol{\mathcal{S}}}^k = \begin{bmatrix} \mathbf{0} \\ (-\boldsymbol{A}^k \boldsymbol{\Phi}_{\boldsymbol{\xi}} + \boldsymbol{\Phi}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}})(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \boldsymbol{\mu}^k_{\boldsymbol{\xi}} \end{bmatrix}$

Reconstruction of GMM from the optimization parameters:

$$\pi^k = e^{\tilde{\pi}^k} / (\textstyle\sum_{i=1}^{K} e^{\tilde{\pi}^i}), \quad \boldsymbol{\Sigma}^k = \boldsymbol{L}^k (\boldsymbol{L}^k)^T$$

Table C.6: Constraints formulation and their derivatives for the alternative Likelihood and MSE cost functions taken from Appendix C.5.

Indices range: $k \in 1..K, \quad c \in 1..d$

Constraint: $\boldsymbol{A}^k + (\boldsymbol{A}^k)^T \prec 0$

The equivalence of the constraint used in the code: $\boldsymbol{\mathcal{C}}_{(k-1)d+c}: \quad (-1)^{c+1} |\boldsymbol{B}_{1:c,1:c}| < 0$

$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \tilde{\pi}^k} = 0$ (valid for both the MSE and Likelihood cost functions)

$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{\mu}^k_i} = 0 \qquad i \in 1..d$ (valid for both the MSE and Likelihood cost functions)

The derivatives specific to the MSE cost function:

$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{L}^k_{ij}} = 0 \qquad i \in 1..d, \quad j \in 1..d$

$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{A}^k_{ij}} = (-1)^{c+1} tr\Big(adj\big(\boldsymbol{B}_{1:c,1:c}\big) \big[\mathbf{0}^{\{\overline{ij}\}}\big]_{1:c,1:c} \Big) \qquad i \in 1..d, \quad j \in 1..d$

The derivative specific to the Likelihood cost function:

$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{L}^k_{ij}} = (-1)^{c+1} tr\Big(adj\big(\boldsymbol{B}_{1:c,1:c}\big) \boldsymbol{\mathcal{X}}_{1:c,1:c} \Big) \qquad i \in 1..2d, \quad j \in 1..i$

where $\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}} (\boldsymbol{L}^k)^T + \boldsymbol{L}^k (\mathbf{0}^{\{ij\}})^T, \ \boldsymbol{\Psi} = \big(-\boldsymbol{A}^k \boldsymbol{\Phi}_{\boldsymbol{\xi}} + \boldsymbol{\Phi}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}\big)(\boldsymbol{\Sigma}_{\boldsymbol{\xi}})^{-1}, \ \boldsymbol{\mathcal{X}} = \boldsymbol{\Psi} + (\boldsymbol{\Psi})^T$

C.1.1 Derivatives w.r.t. π^k

$$\frac{\partial J}{\partial \pi^k} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \pi^k} \qquad \forall k \in 1..K \tag{C.3}$$

The partial derivatives $\frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}}$ and $\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \pi^k}$ can be computed from Eqs. (C.4) and (C.5), respectively:

$$\frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} = \frac{1}{\mathcal{T}} (\dot{\boldsymbol{\xi}}^{t,n} - \boldsymbol{\xi}^{t,n})^T \tag{C.4}$$

$$\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \pi^k} = \frac{h^k(\boldsymbol{\xi}^{t,n})}{\pi^k} (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) \tag{C.5}$$

Substituting Eqs. (C.4) and (C.5) into Eq. (C.3) yields:

$$\frac{\partial J}{\partial \pi^k} = \frac{1}{\pi^k \mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} h^k(\boldsymbol{\xi}^{t,n}) (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) \tag{C.6}$$

C.1.2 Derivatives w.r.t. $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$

Since $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$ is a d-dimensional vector, we need to compute the derivative w.r.t. each component of $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$ separately:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} \quad \forall i \in 1..d,\ k = 1..K \tag{C.7}$$

The partial derivative $\frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}}$ is given by Eq. (C.4), and $\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k}$ is:

$$\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k} = h^k(\boldsymbol{\xi}^{t,n}) \left((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}_{\boldsymbol{\xi}}^k)^T (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \mathbf{0}^{\{i\}} \right) (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) \tag{C.8}$$

where $\mathbf{0}^{\{i\}}$ has the dimension of d.

C.1.3 Derivatives w.r.t. $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$

By substituting directly the constraint Eq. (C.2a) into Eq. (4.10), the partial derivative $\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\mu}_{\dot{\boldsymbol{\xi}},i}^k}$ is always zero because $\boldsymbol{f}(\boldsymbol{\xi})$ no longer depends on $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$. Therefore, $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}},i}^k$ can be dropped from the list of the optimization parameters. In fact, at each iteration $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$ is exploited to satisfy this constraint, and its value can be directly computed from Eq. (C.2a).

C.1.4 Derivatives w.r.t. $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k$

Since $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k$ is a $d \times d$ matrix, we will compute the derivative w.r.t. its each component separately. Since $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k$ is a symmetric matrix, we compute the derivatives only for the components on the lower triangle matrix.

$$\frac{\partial J}{\partial \boldsymbol{\Sigma}^k_{\boldsymbol{\xi},ij}} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\Sigma}^k_{\boldsymbol{\xi},ij}} \qquad \begin{cases} \forall i \in 1..d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.9}$$

The partial derivative $\partial \dot{\boldsymbol{\xi}}^{t,n} / \partial \boldsymbol{\Sigma}^k_{\boldsymbol{\xi},ij}$ is:

$$\begin{aligned} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\Sigma}^k_{\boldsymbol{\xi},ij}} = & -h^k(\boldsymbol{\xi}^{t,n}) \boldsymbol{A}^k \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \boldsymbol{\xi}^{t,n} + \cdots \\ & \frac{h^k(\boldsymbol{\xi}^{t,n})}{2} \Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k_{\boldsymbol{\xi}})^T (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k_{\boldsymbol{\xi}}) + \cdots \\ & - tr\big((\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \mathbf{0}^{\{\overline{ij}\}} \big) \Big) (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) \end{aligned} \tag{C.10}$$

where $\mathbf{0}^{\{\overline{ij}\}}$ has the dimension of $d \times d$.

C.1.5 Derivatives w.r.t. $\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}}}$

The partial derivatives of the cost function w.r.t. the components of $\boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}}}$ are

$$\frac{\partial J}{\partial \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}},ij}} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}},ij}} \qquad \begin{cases} \forall i \in 1..d \\ \forall j \in 1..d \\ \forall k \in 1..K \end{cases} \tag{C.11}$$

The partial derivative $\partial \dot{\boldsymbol{\xi}}^{t,n} / \partial \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}},ij}$ is:

$$\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{\Sigma}^k_{\dot{\boldsymbol{\xi}}\dot{\boldsymbol{\xi}},ij}} = h^k(\boldsymbol{\xi}^{t,n}) \mathbf{0}^{\{ij\}} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \boldsymbol{\xi}^{t,n} \tag{C.12}$$

where $\mathbf{0}^{\{ij\}}$ has the dimension of $d \times d$.

C.2 Alternative MSE Optimization

Though the MSE optimization provided in Appendix C.1 is sufficient to estimate a stable DS, its performance can be significantly increased through a change of optimization parameters. Let us define:

$$\begin{cases} \tilde{\pi}^k = ln(\pi^k) \\ \boldsymbol{L}^k_{\boldsymbol{\xi}} = Chol(\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}}) \end{cases} \tag{C.13}$$

where $\boldsymbol{L}^k_{\boldsymbol{\xi}}$ is a $d \times d$ lower triangle matrix. Since $\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}}$ are positive definite matrix, their Cholesky decomposition $\boldsymbol{L}^k_{\boldsymbol{\xi}}$ always exist. Furthermore, as it was pointed

out before, by substituting Eq. (C.2a) into Eq. (4.10), we can define the evolution of motion with:

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(\boldsymbol{\xi}) = \sum_{k=1}^{K} h^k(\boldsymbol{\xi}) \boldsymbol{A}^k (\boldsymbol{\xi} - \boldsymbol{\xi}^*) \tag{C.14}$$

Considering Appendix C.2 and Eq. (C.14) and defining the optimization parameters to be $\boldsymbol{\theta} = \{\tilde{\pi}^1..\tilde{\pi}^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{L}_{\boldsymbol{\xi}}^1..\boldsymbol{L}_{\boldsymbol{\xi}}^K; \boldsymbol{A}^1..\boldsymbol{A}^K\}$, the alternative MSE optimization can be expressed as:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} (\dot{\boldsymbol{\xi}}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n})^T (\dot{\boldsymbol{\xi}}^{t,n} - -\dot{\boldsymbol{\xi}}^{t,n}) \tag{C.15}$$

subject to

$$\boldsymbol{A}^k + (\boldsymbol{A}^k)^T \prec 0 \qquad \forall k \in 1..K \tag{C.16}$$

where $\dot{\boldsymbol{\xi}}^{t,n} = \boldsymbol{f}(\boldsymbol{\xi}^{t,n})$ are computed from Eq. (C.14). Once the optimization finished, the parameters of GMM can be reconstructed as follows:

$$\begin{cases} \pi^k = e^{\tilde{\pi}^k} / (\sum_{i=1}^{K} e^{\tilde{\pi}^i}) \\ \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k = \boldsymbol{L}_{\boldsymbol{\xi}}^k (\boldsymbol{L}_{\boldsymbol{\xi}}^k)^T \\ \boldsymbol{\Sigma}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^k = \boldsymbol{A}^k \boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k \end{cases} \tag{C.17}$$

In fact the proposed change of parameters allows us to automatically satisfy the last three optimization constraints of Eq. (C.2). The first constraint of Eq. (C.2) is also removed since it is directly considered in Eq. (C.14). The derivatives of the new optimization problem are provided in the following subsections.

C.2.1 Derivatives w.r.t. π^k

$$\frac{\partial J}{\partial \tilde{\pi}^k} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \pi^k} \frac{\partial \pi^k}{\partial \tilde{\pi}^k} \qquad \forall k \in 1..K \tag{C.18}$$

The partial derivatives $\partial J / \partial \dot{\boldsymbol{\xi}}^{t,n}$ and $\partial \dot{\boldsymbol{\xi}}^{t,n} / \partial \pi^k$ are given by Eqs. (C.4) and (C.5), and the derivative $\partial \pi^k / \partial \tilde{\pi}^k$ is simply:

$$\frac{\partial \pi^k}{\partial \tilde{\pi}^k} = e^{\tilde{\pi}^k} \tag{C.19}$$

C.2.2 Derivatives w.r.t. $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$

These derivative can be similarly computed from Eq. (C.7).

C.2.3 Derivatives w.r.t. $\boldsymbol{L}^k$

$\boldsymbol{L}^k_{\boldsymbol{\xi}}$ is a $d \times d$ lower triangle matrix. The partial derivatives of the cost function w.r.t. its parameters are:

$$\frac{\partial J}{\partial \boldsymbol{L}^k_{\boldsymbol{\xi},ij}} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{L}^k_{\boldsymbol{\xi},ij}} \qquad \begin{cases} \forall i \in 1..d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.20}$$

The partial derivative $\partial \dot{\boldsymbol{\xi}}^{t,n} / \partial \boldsymbol{L}^k_{\boldsymbol{\xi},ij}$ is:

$$\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{L}^k_{\boldsymbol{\xi},ij}} = \frac{h^k(\boldsymbol{\xi}^{t,n})}{2} \Bigg((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k_{\boldsymbol{\xi}})^T (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \boldsymbol{\Phi} (\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k_{\boldsymbol{\xi}}) \cdots \\ - tr\Big((\boldsymbol{\Sigma}^k_{\boldsymbol{\xi}})^{-1} \boldsymbol{\Phi} \Big) \Bigg) (\boldsymbol{A}^k \boldsymbol{\xi}^{t,n} - \dot{\boldsymbol{\xi}}^{t,n}) \tag{C.21}$$

where $\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}} (\boldsymbol{L}^k_{\boldsymbol{\xi}})^T + \boldsymbol{L}^k_{\boldsymbol{\xi}} (\mathbf{0}^{\{ij\}})^T$, and has the dimension of $d \times d$..

C.2.4 Derivatives w.r.t. $\boldsymbol{A}^k$

The partial derivatives of the cost function w.r.t. the components of $\boldsymbol{A}^k$ are

$$\frac{\partial J}{\partial \boldsymbol{A}^k_{ij}} = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \dot{\boldsymbol{\xi}}^{t,n}} \frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{A}^k_{ij}} \qquad \begin{cases} \forall i \in 1..d \\ \forall j \in 1..d \\ \forall k \in 1..K \end{cases} \tag{C.22}$$

The partial derivative $\partial \dot{\boldsymbol{\xi}}^{t,n} / \partial \boldsymbol{A}^k_{ij}$ is:

$$\frac{\partial \dot{\boldsymbol{\xi}}^{t,n}}{\partial \boldsymbol{A}^k_{ij}} = h^k(\boldsymbol{\xi}^{t,n}) \mathbf{0}^{\{ij\}} \boldsymbol{\xi}^{t,n} \tag{C.23}$$

where $\mathbf{0}^{\{ij\}}$ has the dimension of $d \times d$.

C.3 Likelihood Optimization

The likelihood optimization is defined as:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \log \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | \boldsymbol{\theta}) \tag{C.24}$$

subject to the same constrains as given by Eq. (C.2). In the above equation, $\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] | \boldsymbol{\theta})$ is given by Eq. (4.6). The optimization parameters for this objective function are: $\boldsymbol{\theta} = \{\pi^1..\pi^K; \boldsymbol{\mu}^1_{\boldsymbol{\xi}}..\boldsymbol{\mu}^K_{\boldsymbol{\xi}}; \boldsymbol{\Sigma}^1..\boldsymbol{\Sigma}^K\}$. Next we compute these derivatives with respect to $\boldsymbol{\theta}$.

C.3.1 Derivatives w.r.t. π^k

$$\frac{\partial J}{\partial \pi^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \mathcal{P}^{t,n}} \frac{\partial \mathcal{P}^{t,n}}{\partial \pi^k} \qquad \forall k \in 1..K \tag{C.25}$$

where for simplicity we shorten the notation $\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]; \boldsymbol{\theta})$ to $\mathcal{P}^{t,n}$. The partial derivatives $\frac{\partial J}{\partial \mathcal{P}^{t,n}}$ and $\frac{\partial \mathcal{P}^{t,n}}{\partial \pi^k}$ can be computed from Eqs. (C.26) and (C.27), respectively:

$$\frac{\partial J}{\partial \mathcal{P}^{t,n}} = -\frac{1}{\mathcal{T}} \frac{1}{\mathcal{P}^{t,n}} \tag{C.26}$$

$$\frac{\partial \mathcal{P}^{t,n}}{\partial \pi^k} = \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k) - \mathcal{P}^{t,n} \tag{C.27}$$

Substituting Eqs. (C.26) and (C.27) into Eq. (C.25) yields:

$$\frac{\partial J}{\partial \pi^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \Big(\frac{\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} - 1 \Big) \tag{C.28}$$

C.3.2 Derivatives w.r.t. $\boldsymbol{\mu}^k_{\boldsymbol{\xi}}$

Special attention should be considered in computing derivatives with respect to $\boldsymbol{\mu}^k_{\boldsymbol{\xi}}$. As it is already discussed in Appendix C.1, there is a direct relation between $\boldsymbol{\mu}^k_{\boldsymbol{\xi}}$ and $\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}$ through the constraint Eq. (C.2a). By substituting the corresponding value of $\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}$ into the cost function given by Eq. (C.24), the optimization no longer depends on $\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}$. Hence, we can drop $\boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}}}$ from the optimization parameters and the constraint Eq. (C.2a) is always satisfied. However, this substitution should be considered when computing the derivatives with respect to $\boldsymbol{\mu}^k_{\boldsymbol{\xi}}$:

$$\frac{\partial J}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \mathcal{P}^{t,n}} \Big(\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} + \sum_{j=1}^{d} \frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},j}} \frac{\partial \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}},j}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} \Big) \tag{C.29}$$

The partial derivative $\frac{\partial J}{\partial \mathcal{P}^{t,n}}$ is given by Eq. (C.26), and $\partial \mathcal{P}^{t,n} / \partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}$ is:

$$\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} = (\mathbf{0}^{\{i\}})^T (\boldsymbol{\Sigma}^k)^{-1} \big([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] - \boldsymbol{\mu}^k \big) \mathcal{P}(k) \mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k) \qquad \forall i \in 1..d \tag{C.30}$$

where $\mathbf{0}^{\{i\}}$ is a vector of dimension $2d$.

The partial derivative $\partial \mathcal{P}^{t,n} / \partial \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}},j}$ can be computed similarly to Eq. (C.29); however by replacing $\mathbf{0}^{\{i\}}$ with $\mathbf{0}^{\{i+d\}}$.

The derivative $\frac{\partial \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}},j}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}}$ can be computed by differentiating Eq. (C.2a) with respect to $\boldsymbol{\mu}^k_{\boldsymbol{\xi},i}$:

$$\frac{\partial \boldsymbol{\mu}^k_{\dot{\boldsymbol{\xi}},j}}{\partial \boldsymbol{\mu}^k_{\boldsymbol{\xi},i}} = \boldsymbol{A}^k_{ji} \qquad \forall i \in 1..d,\ j \in 1..d \tag{C.31}$$

Thanks to matrix multiplication, we can significantly simplify the multiplications by substituting Eqs. (C.26), (C.30) and (C.31) into Eq. (C.29), and compute $\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi}}^k}$:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi}}^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} \begin{bmatrix} \boldsymbol{I} & (\boldsymbol{A}^k)^T \end{bmatrix} (\boldsymbol{\Sigma}^k)^{-1} \left([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}] - \boldsymbol{\mu}^k\right) \quad \forall i \in 1..d \tag{C.32}$$

where $\boldsymbol{I}$ has the dimension of $d \times d$. Note that $\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi}}^k}$ is now a *vector* of dimension d, and each $\frac{\partial J}{\partial \boldsymbol{\mu}_{\boldsymbol{\xi},i}^k}$ is in fact one element of this vector.

C.3.3 Derivatives w.r.t. $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$

By substituting directly the constraint Eq. (C.2a) into Eq. (4.10), the partial derivative $\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\mu}_{\dot{\boldsymbol{\xi}},i}^k}$ is always zero because $\boldsymbol{f}(\boldsymbol{\xi})$ no longer depends on $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$. Therefore, $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$ can be dropped from the list of the optimization parameters. For more information see Appendix C.3.2.

C.3.4 Derivatives w.r.t. $\boldsymbol{\Sigma}^k$

Similar to Appendix C.3.2, we need to consider the effect of substitution of $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$ when computing the derivatives of $\boldsymbol{\Sigma}^k$. All $\boldsymbol{\Sigma}^k$ are $2d \times 2d$ symmetric matrices, hence we compute the derivatives only for the components on the lower triangle matrix.

$$\frac{\partial J}{\partial \boldsymbol{\Sigma}_{ij}^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \mathcal{P}^{t,n}} \left(\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k} + \frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k}\bigg|_{\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k} \right) \qquad \begin{cases} \forall i \in 1..2d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.33}$$

where $\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k}\Big|_{\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k}$ corresponds to the portion of derivatives due to the effect of $\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k$, and can be computed from:

$$\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k}\bigg|_{\boldsymbol{\mu}_{\dot{\boldsymbol{\xi}}}^k} = \sum_{l=1}^{d} \sum_{m=1}^{d} \frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\mu}_{\dot{\boldsymbol{\xi}},l}^k} \frac{\partial \boldsymbol{\mu}_{\dot{\boldsymbol{\xi}},l}^k}{\partial \boldsymbol{A}_{lm}^k} \frac{\partial \boldsymbol{A}_{lm}^k}{\partial \boldsymbol{\Sigma}_{ij}^k} \tag{C.34}$$

The partial derivative $\partial \mathcal{P}^{t,n} / \partial \boldsymbol{\Sigma}_{ij}^k$ is:

$$\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k} = 0.5\Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}} (\boldsymbol{\Sigma}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k) \cdots - tr\left((\boldsymbol{\Sigma}^k)^{-1} \mathbf{0}^{\{\overline{ij}\}}\right) \Big) \mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k) \tag{C.35}$$

where $\mathbf{0}^{\{\overline{ij}\}}$ has the dimension of $2d \times 2d$.

The partial derivative $\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k}\Big|_{\boldsymbol{\mu}_{\boldsymbol{\xi}}^k}$ could be significantly simplified if it is computed in the matrix form (because we can drop the both summations on l and m):

$$\frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{\Sigma}_{ij}^k}\bigg|_{\boldsymbol{\mu}_{\boldsymbol{\xi}}^k} = \mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T(\boldsymbol{\Sigma}^k)^{-1}\boldsymbol{\mathcal{S}}^k \tag{C.36}$$

where $\boldsymbol{\mathcal{S}}^k$ is a vector of dimension $2d$ and is equal to:

$$\boldsymbol{\mathcal{S}}^k = \begin{bmatrix} \mathbf{0} \\ \Big(- \boldsymbol{A}^k\big[\mathbf{0}^{\{\overline{ij}\}}\big]_{\boldsymbol{\xi}} + \big[\mathbf{0}^{\{\overline{ij}\}}\big]_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}\Big)(\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1}\boldsymbol{\mu}_{\boldsymbol{\xi}}^k \end{bmatrix} \tag{C.37}$$

In Eq. (C.37), $\mathbf{0}$ is a zero column vector of dimension d, and $\mathbf{0}_{\boldsymbol{\xi}}^{\{\overline{ij}\}}$ and $\mathbf{0}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}^{\{\overline{ij}\}}$ are partitions of $\mathbf{0}^{\{\overline{ij}\}}$. Finally, by substituting Eqs. (C.26), (C.35) and (C.36) into Eq. (C.33) we have:

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{\Sigma}_{ij}^k} = -\frac{1}{2\mathcal{T}}\sum_{n=1}^{N}\sum_{t=0}^{T^n}\frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}}\Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T(\boldsymbol{\Sigma}^k)^{-1}\mathbf{0}^{\{\overline{ij}\}}(\boldsymbol{\Sigma}^k)^{-1}(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k) \\ - tr\big((\boldsymbol{\Sigma}^k)^{-1}\mathbf{0}^{\{\overline{ij}\}}\big) + 2(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T(\boldsymbol{\Sigma}^k)^{-1}\boldsymbol{\mathcal{S}}^k\Big)\end{aligned} \tag{C.38}$$

C.4 Alternative Likelihood Optimization

Similarly to Appendix C.2, we can define an alternative likelihood optimization so that 4 out of 5 optimization constraints can be automatically satisfied through a change of variable:

$$\begin{cases}\tilde{\pi}^k = ln(\pi^k) \\ \boldsymbol{L}^k = Chol(\boldsymbol{\Sigma}^k)\end{cases} \tag{C.39}$$

where $\boldsymbol{L}^k$ are $2d \times 2d$ lower triangle matrices. Since $\boldsymbol{\Sigma}^k$ are positive definite matrices, their Cholesky decomposition always exist. The alternative likelihood optimization can be expressed as:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = -\frac{1}{\mathcal{T}}\sum_{n=1}^{N}\sum_{t=0}^{T^n}\log\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|\boldsymbol{\theta}) \tag{C.40}$$

subject to

$$\boldsymbol{A}^k + (\boldsymbol{A}^k)^T \prec 0 \qquad \forall k \in 1..K \tag{C.41}$$

where $\boldsymbol{\theta} = \{\tilde{\pi}^1..\tilde{\pi}^K; \boldsymbol{\mu}_{\boldsymbol{\xi}}^1..\boldsymbol{\mu}_{\boldsymbol{\xi}}^K; \boldsymbol{L}^1..\boldsymbol{L}^K\}$. Once the optimization finished, the pa-

rameters of GMM can be reconstructed as follows:

$$\begin{cases} \pi^k = e^{\tilde{\pi}^k} / (\sum_{i=1}^{K} e^{\tilde{\pi}^i}) \\ \boldsymbol{\Sigma}^k = \boldsymbol{L}^k (\boldsymbol{L}^k)^T \end{cases} \tag{C.42}$$

In fact the proposed change of parameters allows us to automatically satisfy the last three optimization constraints of Eq. (C.2). The first constraint of Eq. (C.2) is also removed since it is directly considered in Eq. (C.14). The derivatives of the new optimization problem are provided in the following subsections.

C.4.1 Derivatives w.r.t. π^k

$$\frac{\partial J}{\partial \tilde{\pi}^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \mathcal{P}^{t,n}} \frac{\partial \mathcal{P}^{t,n}}{\partial \pi^k} \frac{\partial \pi^k}{\partial \tilde{\pi}^k} \qquad \forall k \in 1..K \tag{C.43}$$

The partial derivatives $\partial J / \partial \mathcal{P}^{t,n}$, $\partial \mathcal{P}^{t,n} / \partial \pi^k$ and $\partial \pi^k / \partial \tilde{\pi}^k$ are given by Eqs. (C.19), (C.26) and (C.27), respectively.

C.4.2 Derivatives w.r.t. $\boldsymbol{\mu}_{\boldsymbol{\xi}}^k$

These derivative can be similarly computed from Eq. (C.29).

C.4.3 Derivatives w.r.t. $\boldsymbol{L}^k$

$\boldsymbol{L}^k$ is a $2d \times 2d$ lower triangle matrix. The partial derivatives of the cost function with respect to the optimization parameters are:

$$\frac{\partial J}{\partial \boldsymbol{L}_{ij}^k} = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\partial J}{\partial \mathcal{P}^{t,n}} \frac{\partial \mathcal{P}^{t,n}}{\partial \boldsymbol{L}_{ij}^k} \qquad \begin{cases} \forall i \in 1..2d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.44}$$

The partial derivative $\partial \mathcal{P}^{t,n} / \partial \boldsymbol{L}_{ij}^k$ is:

$$\frac{\partial J}{\partial \boldsymbol{L}_{ij}^k} = -\frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \frac{\mathcal{P}(k)\mathcal{P}([\boldsymbol{\xi}^{t,n}; \dot{\boldsymbol{\xi}}^{t,n}]|k)}{\mathcal{P}^{t,n}} \Big((\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \boldsymbol{\Phi} (\boldsymbol{\Sigma}^k)^{-1} (\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k) - tr\big((\boldsymbol{\Sigma}^k)^{-1} \boldsymbol{\Phi}\big) + 2(\boldsymbol{\xi}^{t,n} - \boldsymbol{\mu}^k)^T (\boldsymbol{\Sigma}^k)^{-1} \tilde{\boldsymbol{\mathcal{S}}}^k \Big) \tag{C.45}$$

where $\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}} (\boldsymbol{L}^k)^T + \boldsymbol{L}^k (\mathbf{0}^{\{ij\}})^T$, and has the dimension of $2d \times 2d$. The $2d$ dimension vector $\tilde{\boldsymbol{\mathcal{S}}}^k$ is:

$$\tilde{\boldsymbol{\mathcal{S}}}^k = \begin{bmatrix} \mathbf{0} \\ \big(-\boldsymbol{A}^k \boldsymbol{\Phi}_{\boldsymbol{\xi}} + \boldsymbol{\Phi}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}} \big) (\boldsymbol{\Sigma}_{\boldsymbol{\xi}}^k)^{-1} \boldsymbol{\mu}_{\boldsymbol{\xi}}^k \end{bmatrix} \tag{C.46}$$

where $\mathbf{0}$ is a zero column vector of dimension d.

C.5 Optimization Constraints and Their Derivatives

In this section we provide formulations for the optimization problems defined in Appendices C.2 and C.4, where the only constraint is the negative definiteness of matrices $\boldsymbol{A}^k$. To ensure this constraint, we first need to define a method to mathematically determine whether a matrix is negative definite. There are several ways to ensure whether a symmetric matrix $\boldsymbol{B}$ is negative definite, among which the two most famous ones are 1) verifying all eigenvalues of $\boldsymbol{B}$ are strictly negative, 2) using Sylvester's criterion. In our work, we use Sylvester's criterion because it provides us with an analytical formulation to verify negative definiteness (compared to computing eigenvalues which is an iterative procedure).

Sylvester's criterion states that a Hermitian matrix $\boldsymbol{B}$ is negative-definite if and only if the determinant of all i-th order leading principal minors[1] are negative if i is odd and positive if i is even (Gilbert, 1991). Each $d \times d$ symmetric matrix has d principal minors. By defining $\boldsymbol{B}^k = \boldsymbol{A}^k + (\boldsymbol{A}^k)^T$, the optimization constraint given by Eq. (C.41) is equal to:

$$\boldsymbol{\mathcal{C}}_{(k-1)d+c}: \quad (-1)^{c+1} \left|\boldsymbol{B}_{1:c,1:c}\right| < 0 \qquad \begin{cases} \forall c \in 1..d \\ \forall k \in 1..K \end{cases} \tag{C.47}$$

where we use $\boldsymbol{\mathcal{C}}_{(k-1)d+c}$ to refer to the $((k-1)d+c)$-th constraint. Thus for a GMM model composed of K Gaussian functions, there are $K \times d$ constraints that should be satisfied during the optimization. The derivative of these constraints with respect to π^k and $\boldsymbol{\mu}^k$ are always zero, irrespective of which cost function is used:

$$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \tilde{\pi}^k} = 0 \qquad \begin{cases} \forall c \in 1..d \\ \forall k \in 1..K \end{cases} \tag{C.48}$$

$$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{\mu}_i^k} = 0 \qquad \begin{cases} \forall c \in 1..d \\ \forall i \in 1..2d \\ \forall k \in 1..K \end{cases} \tag{C.49}$$

For the MSE optimization defined by Appendix C.2 we have:

$$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{L}_{ij}^k} = 0 \qquad \begin{cases} \forall c \in 1..d \\ \forall i \in 1..d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.50}$$

[1] The i-th principal minor of a $d \times d$ symmetric matrix B is a quadratic upper-left part of $\boldsymbol{B}$, which consists of matrix elements in rows and columns from 1 to d.

$$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{A}_{ij}^{k}} = (-1)^{c+1} tr\Big(adj\big(\boldsymbol{B}_{1:c,1:c}\big)\big[\mathbf{0}^{\{\overline{ij}\}}\big]_{1:c,1:c}\Big) \quad \begin{cases} \forall c \in 1..d \\ \forall i \in 1..d \\ \forall j \in 1..d \\ \forall k \in 1..K \end{cases} \tag{C.51}$$

where $\mathbf{0}^{\{\overline{ij}\}}$ has the dimension of $d \times d$. For the likelihood optimization defined by Appendix C.4 we have:

$$\frac{\partial \boldsymbol{\mathcal{C}}_{(k-1)d+c}}{\partial \boldsymbol{L}_{ij}^{k}} = (-1)^{c+1} tr\Big(adj\big(\boldsymbol{B}_{1:c,1:c}\big)\boldsymbol{\mathcal{X}}_{1:c,1:c}\Big) \quad \begin{cases} \forall c \in 1..d \\ \forall i \in 1..2d \\ \forall j \in 1..i \\ \forall k \in 1..K \end{cases} \tag{C.52}$$

where $\boldsymbol{\mathcal{X}}$ is a $d \times d$ symmetric matrix defined by:

$$\boldsymbol{\Phi} = \mathbf{0}^{\{ij\}}(\boldsymbol{L}^{k})^{T} + \boldsymbol{L}^{k}(\mathbf{0}^{\{ij\}})^{T} \tag{C.53}$$

$$\boldsymbol{\Psi} = \big(-\boldsymbol{A}^{k}\boldsymbol{\Phi}_{\boldsymbol{\xi}} + \boldsymbol{\Phi}_{\dot{\boldsymbol{\xi}}\boldsymbol{\xi}}\big)(\boldsymbol{\Sigma}_{\boldsymbol{\xi}})^{-1} \tag{C.54}$$

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\Psi} + (\boldsymbol{\Psi})^{T} \tag{C.55}$$

where $\boldsymbol{\Phi}$ and $\mathbf{0}^{\{ij\}}$ are $2d \times 2d$ matrices, and $\boldsymbol{\Psi}$ is a $d \times d$ matrix.

Appendix D

Learning Hitting Parameters in Minigolf

As outlined in Chapter 5, the minigolf task requires two skills: 1) A default hitting motion $\boldsymbol{f}_h(\boldsymbol{\xi})$ that can generate motions from different initial positions and that can be altered in terms of the hitting parameters, and 2) A field-specific estimate of a mapping from input space to the hitting parameters $(\alpha, \kappa) = \boldsymbol{g}(\boldsymbol{s})$ that defines what hitting parameters should be used for each situation.

The first part has already been covered in Chapter 5. This appendix reports on the second problem, and discusses the problem of learning the hitting parameters (angle and speed with which to hit the ball) from a training set collected with the aid of a teacher specifying good values for some different hitting locations. This method is evaluated on two challenging fields using the 7-DoF Barrett WAM arm. The material presented in this appendix was done in close collaboration with Klas Kronander, during his master thesis, under my advice, and beginning of PhD thesis at LASA.

D.1 Hitting Parameters

After learning an adaptable hitting motion that can be used to hit with different speed and direction, the robot needs to learn what speed and direction should be used for each situation, i.e. which κ and α should be generated for each input vector $\boldsymbol{s}$. Furthermore there is generally more than one valid combination of hitting parameters for each input point on advanced fields. In this section, we refer to these different possibilities of choosing the hitting parameters as *strategies*.

Fig. D.1 shows samples of two strategies for one ball location for an arctan-shaped field. While learning all the strategies for a field certainly gives the player more freedom to vary her game, mastering one strategy should be sufficient for a successful game. By assuming that a strategy can be represented by a continuous mapping from the relative position of the ball and the hole to the hitting parameters, the problem is reduced to estimating this mapping:

$$\boldsymbol{g} : \boldsymbol{s} \mapsto (\alpha, \kappa) \tag{D.1}$$

To learn $\boldsymbol{g}$, we take a supervised learning approach and provide a training set of good parameters for different inputs. Note that the training data is field-specific, as each field requires different hitting parameters.

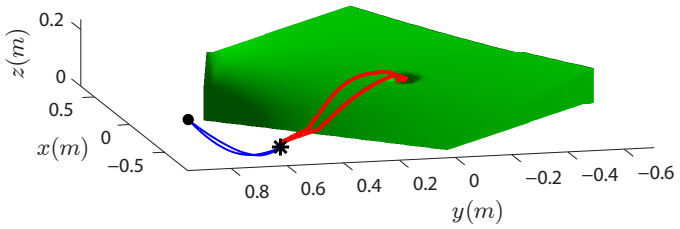
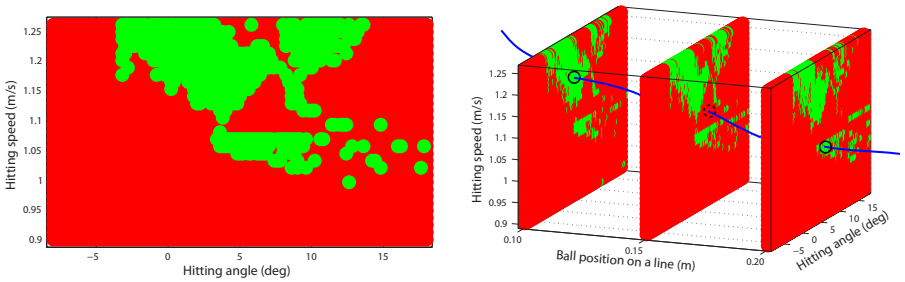


Figure D.1: The figure illustrates a typical situation for advanced fields: for a given relative position between the ball and the hole, there are several combinations of hitting speed and hitting angle that will lead to sinking the ball. The two ball trajectories are represented by the red lines. The starting point, trajectory and impact point of the end-effector are represented by black circle, blue line and black star, respectively. Two different strategies are applied in this figure, one with a high hitting speed and a less curved trajectory, and one where compensating for the fields slope by launching the ball at a bigger angle yet lower hitting speed.



(a) **(b)**

Figure D.2: **(a)** This graph shows the successful (green) or unsuccessful (red) result when using the corresponding hitting parameters for a particular ball position on the arctan field. Several strategies are clearly distinguishable. **(b)** This graph illustrates the problem of picking training data from different strategies. The test point in the middle will average the two encircled training points on the left and right ball positions, resulting in the dashed encircled hitting parameters and thus failing to sink the ball.

D.1.1 Training Data

As outlined above, the problem of estimating the hitting parameters based on the situation on the field is a redundant problem. There are several different strategies a player can choose from when deciding how to hit the ball. Note that within each strategy, there is a range of different angles and speeds that leads to sinking the ball, due to the fact that the hole is larger than the ball. Strategies are often represented by distinguishable separated sets of hitting parameters combinations (see Fig. D.2a). Consequently, using training samples from different strategies to infer hitting parameters for new inputs will generally fail. This is illustrated in Fig. D.2b. The acceptable error margins within each strategy vary in a nonlinear manner across the input space, and it is therefore not useful to determine a bound for the acceptable predictive error, as such a bound would have to be unnecessarily strict for most points to comply with the demands of the points were the acceptable error margin is small.

Consider a set of M observations of good examples[1] $\{\boldsymbol{s}^m, \alpha^m, \kappa^m\}_{m=1}^M$. Following the assumption that we are looking for a function $(\alpha, \kappa) = \boldsymbol{g}(\boldsymbol{s})$, we assume that the training set consists of noisy observations of this function[2]:

$$\{\boldsymbol{s}^m, \alpha^m, \kappa^m\}_{m=1}^M = \{\boldsymbol{s}^m, g_\alpha(\boldsymbol{s}^m) + \epsilon_\alpha, g_\kappa(\boldsymbol{s}^m) + \epsilon_\kappa\}_{m=1}^M \tag{D.2}$$

with noise ϵ_α and ϵ_κ corrupting the angle and speed part, respectively. For clarity, we introduce the following notation used specifically for the training data:

$$\{\boldsymbol{S}, \boldsymbol{\alpha}, \boldsymbol{\kappa}\} = \{\boldsymbol{s}^m, g_\alpha(\boldsymbol{s}^m) + \epsilon_\alpha, g_\kappa(\boldsymbol{s}^m) + \epsilon_\kappa\}_{m=1}^M \tag{D.3}$$

D.1.2 Hitting Parameters Prediction

In this work, we use two different statistical methods to infer the hitting parameters for new inputs using the training set specified above. We provide a recap of these methods here. For more detail information, refer to Section 2.2.

Consider now the mapping in Eq. (5.2). We assume that this mapping is drawn from a distribution over functions defined by a Gaussian Process (GP) fully specified by its covariance function. This assumption implies any set of samples from this function have a joint Gaussian distribution. For any test point $\boldsymbol{s}^*$, the GPR with estimate $\hat{g}_\alpha(\boldsymbol{s}^*)$ and the predictive variance $\boldsymbol{\Sigma}_\alpha^*(\boldsymbol{s}^*)$ can be obtained by conditioning the multivariate Gaussian distribution on the training data:

$$\hat{g}_\alpha(\boldsymbol{s}^*) = \mathbf{K}_\alpha(\boldsymbol{s}^*, \boldsymbol{S})(\mathbf{K}_\alpha(\boldsymbol{S}, \boldsymbol{S}) + \sigma_n \mathbf{I})^{-1}\boldsymbol{\alpha} \tag{D.4a}$$

$$\boldsymbol{\Sigma}_\alpha^*(\boldsymbol{s}^*) = \mathbf{K}_\alpha(\boldsymbol{s}^*, \boldsymbol{s}^*) - \mathbf{K}_\alpha(\boldsymbol{s}^*, \boldsymbol{S})(\mathbf{K}_\alpha(\boldsymbol{S}, \boldsymbol{S}))^{-1}\mathbf{K}_\alpha(\boldsymbol{S}, \boldsymbol{s}^*) \tag{D.4b}$$

The symmetric matrices $\mathbf{K}$ above represent the evaluation of the GP covariance function across the specified variables. We use a squared exponential with different length scales for the different dimensions in input space:

$$k(\boldsymbol{s}, \boldsymbol{s}') = \sigma e^{-(\boldsymbol{s}-\boldsymbol{s}')^T \mathbf{L}(\boldsymbol{s}-\boldsymbol{s}')} \qquad \text{with} \qquad \mathbf{L} = \begin{pmatrix} l_1 & 0 \\ 0 & l_2 \end{pmatrix}$$

The scalars l_1 and l_2 are the length scales of the covariance function. The scalar σ is the signal variance. We use a conjugate-gradient based search algorithm available in GPML[3] for optimizing these hyper-parameters for maximum likelihood of the training set. The above equations also apply to the hitting speed g_κ, with replacement of α and $\boldsymbol{\alpha}$ with κ and $\boldsymbol{\kappa}$, respectively[4].

[1] Note that these examples are *not* the same as the demonstrations of the default hitting motion.

[2] The noise on the observations represents the small redundancies caused by the hole being larger than the ball.

[3] GPML is a Matlab toolbox for GPR, written by C.E. Rasmussen and H. Nickisch.

[4] The parameters of the covariance function are also different, since these are optimized for each data set.

Another way to infer the hitting parameters for new situations is to fit a GMM to the training set. By conditioning the GMM on new query points, the corresponding hitting parameters are inferred. Given the number of Gaussian functions K, the parameters of GMM can be optimized to maximize the likelihood of the training set. In this work, we first cluster the data using k-means and then apply the EM algorithm to optimize the parameters. Then, GMR is used to find hitting parameters for unseen inputs:

$$\hat{\boldsymbol{g}}(\boldsymbol{s}^*) = \sum_{k=1}^{K_{HP}} h_{HP}^k(\boldsymbol{s}) \left(\boldsymbol{\Sigma}_{HP,\alpha\kappa\boldsymbol{s}}^k (\boldsymbol{\Sigma}_{HP,\boldsymbol{s}}^k)^{-1} (\boldsymbol{s} - \boldsymbol{\mu}_{HP,\boldsymbol{s}}^k) + \boldsymbol{\mu}_{HP,\alpha\kappa}^k\right) \tag{D.5}$$

where the nonlinear weighting $h_{HP}^k(\boldsymbol{s})$ is computed in the same way as described by Eq. (4.9). The subscript HP for Hitting Parameters is used above to distinguish the above GMM from those that are used in the reproduction of the hitting motion.

Note that here, we are predicting both the hitting speed and angle by using a joint probability distribution over the input data and both hitting parameters. Thus, in contrast to using GPR where each parameter is predicted independently of the other, when using the GMM we take the dependency across the hitting parameters into account. Similarly, separate GMM can be built encoding the demonstrated $\{\boldsymbol{S}, \boldsymbol{\alpha}\}$ and $\{\boldsymbol{S}, \boldsymbol{\kappa}\}$ to perform GMR where the hitting parameters are predicted independently.

While GPR and GMR are both powerful methods widely used in robotics, they have some important differences in characteristics that affect how well they perform in the context of predicting hitting parameters. Consider first a flat field, as in Fig. 5.1b. For this field, the mapping of hitting parameters has low complexity, and a pattern observed from training data is likely valid outside the training range. As GPR is based on correlation related to the distance in input space, it outputs zero far from the training data. GMR on the other hand, has better generalization ability in that the model extends further outside the training range. Low complexity fields typically also are not very sensitive to errors in hitting parameters, i.e the precision is less important than for advanced fields. For more advanced fields such as the arctan field in Fig. 5.1c, higher precision is required as well as greater flexibility to capture local patterns. GPR has better local precision than GMR, which means that it should outperform GMR for advanced fields where high precision is required when selecting the hitting parameters.

D.2 Evaluation of Hitting Parameters

We evaluated the performance of our system to predict the different hitting parameters on a 7-DoF Barrett arm manipulator. The experiments on the real robot were performed on two fields: a rough flat field, and a field with two hills. The latter will be referred to as the double hill field. Model of these fields

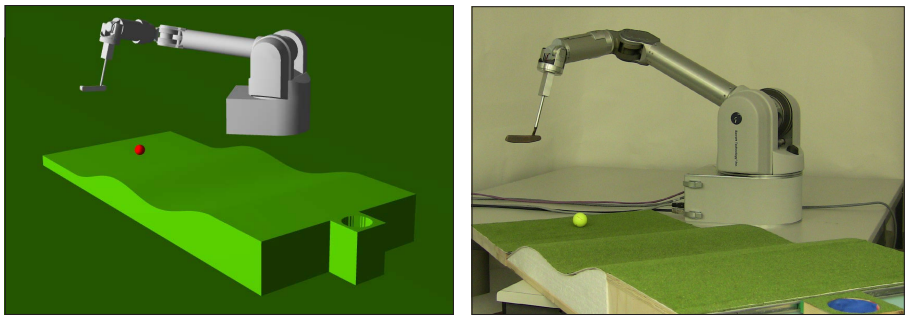


Figure D.3: The double hill field in simulator (left) and with real robot (right).

were used for experiments in a simulated environment using RobotToolKit, see Fig. D.3. In addition to these fields, the arctan field (see Fig. D.1) was used in the simulator. Kinesthetic demonstrations from the real robot were used to learn a hitting motion model which was then used both in the simulator and on the real robot.

The minigolf playing robot uses Eq. (5.9) with κ and α specified either 1) By the teacher during collection of training set for hitting parameters adaptation, or 2) By the trained models presented in Appendix D.1 during autonomous task reproduction. In our experiments, the hitting motion was executed by first transferring the output from Eq. (5.9) and the end-effector orientation to joint space using the damped least squares pseudo-inverse kinematics. Then these values were converted into motor commands using an inverse dynamics controller. Both steps were carried out in realtime at 500Hz.

D.2.1 Results from the Robot Simulation

In the first robot experiment, data sets consisting of 20 points were collected along one dimension in input space of the flat and the double hill fields. In practice, the input dimension was changed by moving the hole sideways along the edge of the field (see Fig. D.3). The strategy was selected by fixing the speed to a constant value for all the hitting attempts. A range of points around the center of the input range, represented by black crosses in Fig. D.4, were selected for training. The results confirm the hypothesis that GMR has a better generalization performance outside the training set, as is clearly visible in Fig. D.4.

Another experiment was centered on comparing the importance of structure when selecting training data. This is an interesting point of comparison, as the teacher might find it non-intuitive to provide training-data with some predefined structures in input-space, e.g. evenly spaced points. The data sets from the preceding experiments were used here as well. For the arctan field, a data set consisting of 56 points was collected. To ensure that all data points were sampled from the same strategy, we chose hitting parameters so as to minimize the hitting speed. This strategy corresponds to the lower of the three green

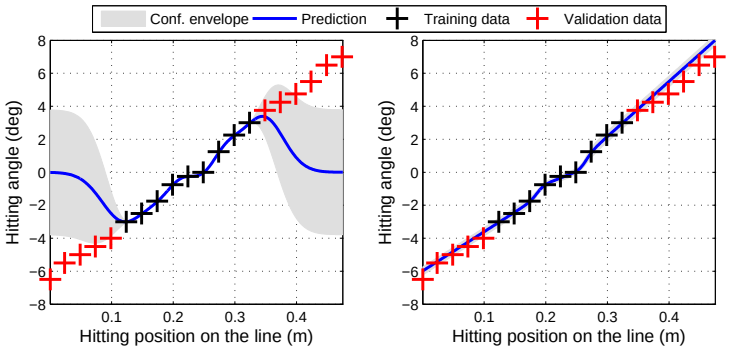


Figure D.4: Red and Black crosses represent a data set of successful hitting angles for the double hill field. The points marked with black crosses were used for regression using GPR (left) and GMR (right). The gray area represents the predictive confidence by two standard deviations ($\sim$ 95 %).

fields representing the main strategies in Fig. D.2. From the different data sets, training points were selected according to Table D.1. The remainder of the data sets were used for validation of the trained models, resulting in the Root Mean Square Error (RMSE) in Table D.1. The rates of success were determined by comparing random predictions for 30 datapoints selected randomly in the ranges of input used. As there are random elements both in the learning phase and more importantly in the training data selection phase, the training data selection and training were carried out 100 times for each case. The values for RMSE and rates of success are the averages of these rollouts.

The results in Table D.1 clearly reveal the difference in sensitivity to the training data for the two methods. Overall GMR performs better than GPR both in terms of precision and rate of success when the training data is selected at random. However, for the evenly spaced training data, GPR clearly takes the lead. This difference is most notable for the arctan field, where the highly complex data set is handled much better by GPR. The advantage for GPR would likely be even bigger for more advanced fields. The reason the algorithms perform worse with randomly selected data is mainly because some regions in input space are likely to be poorly represented in the selected data set. Thus, there are simply no examples to learn from in these regions.

Furthermore, the results of this experiment indicate the higher performance of the joint GMM model versus the separate GMMs. By training one model for both hitting parameters, higher performance was achieved while using fewer parameters. In contrast to the separate GMMs, the joint GMM models the correlation between the hitting parameters. This additional information, available when training the joint GMM but not when training the separate GMMs, could possibly explain the increase in performance. The correlation is illustrated in Fig. D.5. Even though we deal with very small data sets here, GMR has an advantage compared to GPR in terms of the number of parameters for all cases except when the smallest data sets are considered. Naturally, the difference in the number of parameters grows with the size of the training set.

Table D.1: Results summary for learning the hitting parameters on data from the robot simulator.

	Model	RMSE angle	RMSE speed	Success rate	No. of parameters
The rough flat field	5 random training points				
	GPR	1.40	-	0.54	19
	GMR	0.52	-	0.64	20
	10 random training points				
	GPR	0.77	-	0.59	34
	GMR	0.35	-	0.79	20
	10 equally spaced training				
	GPR	0.15	-	0.96	34
	GMR	0.23	-	0.94	20
The Double hill field	5 random training points				
	GPR	1.60	-	0.36	19
	GMR	0.80	-	0.43	20
	10 random training points				
	GPR	1.41	-	0.42	34
	GMR	0.35	-	0.52	20
	10 equally spaced training				
	GPR	0.18	-	0.87	34
	GMR	0.27	-	0.83	20
The arctan field	16 random training points				
	GPR	0.94	0.02	0.85	72
	Separate GMR	1.01	0.02	0.86	60
	GMR	1.01	0.02	0.87	45
	28 equally spaced training				
	GPR	0.24	0.01	0.96	120
	Separate GMR	0.52	0.02	0.88	60
	GMR	0.95	0.02	0.93	45

D.2.2 Results from the Real Robot

The promising results from the robot simulator were confirmed on the real robot, using the rough flat and the double hill fields. Similarly to the simulator data sets, 20 points of successful input-parameter combinations were collected. The speed was fixed. A higher complexity was expected from these data set compared to their simulator counterparts, as a number of issues were not included in the simulator models, e.g. the dimples on the golf ball and the structure of the artificial grass covering the fields. Indeed, the data sets were more complex, which is reflected in Table D.2, as the learning (with the same methods and parameters) yielded models poorer than those that were learned from the simulator data sets in almost all cases. When the models were trained, the hole was moved to a random location along the slider on the edge of the field (see Fig. D.6). The location of the hole was captured by a stereo vision system operating at 80Hz, allowing the hitting parameters to continuously be updated to the current position of the hole. Thirty points were tested to determine the rate of success. Note that for the double hill we considered an upward circular shaped resting motion to avoid hitting the field.

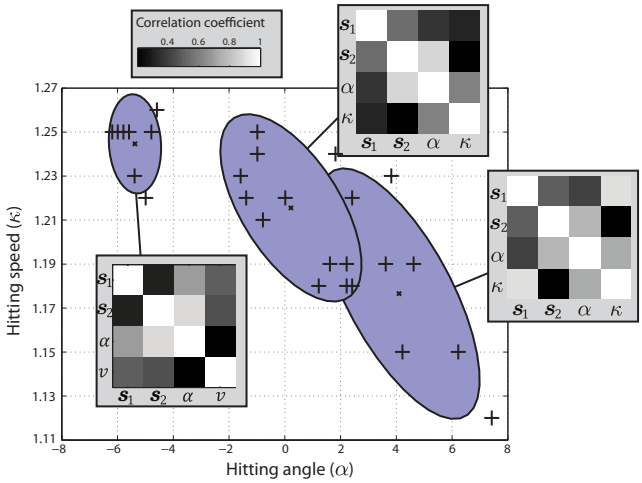


Figure D.5: The figure shows the two output dimension of a GMM with 3 components fitted to a data set from the arctan field. The input dimension $\boldsymbol{s} \in \mathbb{R}^2$ has been marginalized out to illustrate the correlation between the hitting speed and the hitting angle. It also illustrates the absolute correlation matrices associated to each of the Gaussian functions. As can be seen, there is very strong correlation between the hitting parameters in two of the components.

Table D.2: Results summary for learning the hitting parameters on data from the Barrett WAM robotic arm.

	Model	RMSE of angle	Success rate	No. of param.
The rough flat field	5 random training points			
	GPR	1.20	0.57	19
	GMR	0.68	0.63	20
	10 random training points			
	GPR	0.78	0.77	34
	GMR	0.55	0.77	20
	10 equally spaced training			
	GPR	0.16	0.97	34
	GMR	0.22	0.90	20

	Model	RMSE of angle	Success rate	No. of param.
The Double hill field	5 random training points			
	GPR	1.88	0.30	19
	GMR	0.72	0.37	20
	10 random training points			
	GPR	1.72	0.30	34
	GMR	0.42	0.40	20
	10 equally spaced training			
	GPR	0.23	0.70	34
	GMR	0.32	0.73	20

D.3 Conclusion and Discussion

In this appendix, we have presented a statistical approach to learn the parameters of the hitting motion for the minigolf. Here we assumed that despite the many options one typically has for hitting the ball, learning one combination of hitting parameters for each input would be sufficient. In choosing this approach, the goal was to build a high performance model using only a small set of training data. These assumptions turned out reasonable, as very successful models were built from small sets of training data collected in the simulator as well as on the real robot. We showed how two different statistical methods can be used to learn the hitting parameters selection, and compared them in terms of performance to predict hitting parameters for the task at hand. It is likely that simpler regression techniques (e.g. linear regression) could be used to predict the parameters for simple fields such as the flat field. In using a more flexible learning algorithm such as GPR or GMR, the system can handle

Figure D.6: The hitting motion on the WAM. The ball and the hole are continuously tracked by a stereovision system.

a wider range of fields without changing the learning algorithm. Also note that by using a nonlinear regression technique, the learning of the hitting parameters can automatically compensate for errors arising from the hitting motion and/or the robot controller[5].

The proposed learning approach for the hitting parameters is able to generalize well from a small set of training data on the field for which the training data was provided. Note that the system is based on demonstrated data only, and does not use any physical model of the field. This has the advantage that the learning problem becomes relatively simple, and the disadvantage that it is not possible to generalize across different fields. A possible extension would be to reuse a basic learned model (e.g. a GMM with one or two Gaussian functions) on new fields. In such a system, the robot could exhibit very basic generalization to new fields, and the teacher could use the output from that model as a first guess when searching for successful hitting parameters.

Throughout this chapter, we have highlighted the importance of choosing training data from the same strategy, as averaging samples taken from different strategies will generally lead to the selection of inappropriate hitting parameters. This high level selection of training data is intuitive to humans. Most of the previous works that deals with situation based adaptation of motions (Kober, Oztop, & Peters, 2010; Nemec et al., 2009) use reinforcement learning for learning to adapt to new situations through trial and error. Applying such an approach to hitting parameters selection in minigolf presents an interesting challenge, since the cost-function must be designed to favor only one strategy.

As mentioned, a significant simplification of the problem was made in learning only one way to hit the ball for each situation. An interesting approach would be to explore and store several successful parameters for each situation, and to cluster them into different strategies. When trained with such a data set, the robot could be programmed to use the strategy most likely to result in a successful attempt at each hitting point.

[5] Provided that these errors are repeatable and present during the demonstration of hitting parameters.

Appendix E

Publications by the Author

This appendix lists the publications of the author during his doctoral studies. The articles are listed according to the date of their publication.

Journal Papers

- S.M. Khansari Zadeh, K. Kronander, and A. Billard (2012), Learning to Play Minigolf: A Dynamical System-based Approach, *Advanced Robotics*, p. 1–27.
- S.M. Khansari Zadeh and A. Billard (2012), A Dynamical System Approach to Realtime Obstacle Avoidance, *Autonomous Robots*, 32(4), p. 433–454.
- S.M. Khansari Zadeh and A. Billard (2011), Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models, *IEEE Transaction on Robotics*, 27(5), p. 943–957.
- E. Gribovskaya, S.M. Khansari Zadeh, and A. Billard (2010), Learning Non-linear Multivariate Dynamics of Motion in Robotic Manipulators, *International Journal of Robotics Research*, 30(1), p. 80–117.

Peer-reviewed proceedings

- K. Kronander, S.M. Khansari Zadeh, and A. Billard (2011), Learning to Control Planar Hitting Motions in a Minigolf-like Task, *In proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 710–717, Received the JTSC Novel Technology Paper Award.
- S.M. Khansari Zadeh and A. Billard (2010), Imitation learning of Globally Stable Non-Linear Point-to-Point Robot Motions using Nonlinear Programming, *In proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 2676–2683.
- S.M. Khansari Zadeh and A. Billard (2010), BM: An Iterative Method to Learn Stable Non-Linear Dynamical Systems with Gaussian Mixture Models, *In proceedings of the International Conference on Robotics and Automation (ICRA)*, p. 2381–2388.

Conference Workshops

- S.M. Khansari Zadeh and A. Billard (2012), Realtime Avoidance of Fast Moving Objects: A Dynamical System-based Approach, *In electronic proceedings of the Workshop on Robot Motion Planning: Online, Reactive, and in Real-Time*, The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

- S.M. Khansari Zadeh and A. Billard (2011), Learning to Play Mini-Golf from Human Demonstration using Autonomous Dynamical Systems, *In electronic proceedings of the Workshop on New Developments in Imitation Learning*, International Conference on Machine Learning (ICML).

- S.M. Khansari Zadeh and A. Billard (2009), Learning and Control of UAV maneuvers Based on Demonstrations, *Presented in the Workshop on Autonomous Flying vehicles: Fundamentals and Applications*, Robotics: Science and Systems.

Appendix F

Student Projects Supervised by the Author

This appendix lists the projects that were supervised by the author in the scope of this thesis. While not all projects resulted in material that could be directly put to use in this thesis, they nevertheless provided useful insights and results. The projects are listed according to their date.

Semester project, *Spring 2012*

Student: Burak Zeydan

Title: Implementation of the whole body collision avoidance on the WAM arm

> Many robot tasks require reaching or placing an object while avoiding collision with other objects as well as the robot itself. The aim of this project is to take the existing code for the obstacle avoidance, that is only applicable to point-robots, and to extend it for the whole robot body collision avoidance. To obtain this, the student needs to find the closest point on the robot to the obstacle, and then drive it away (with the help of kinematic null-space). In this project, the following steps should be obtained: 1) To understand the existing C++ code and the approach, 2) to implement the closest point on the robot to the obstacle, and 3) To drive the obtained point away from the obstacle.

Semester project[1], *Winter 2011*

Student: Adrien Béraud

Title: Development of a Control Module for Learning Robot Minigolf Hitting Motions on Advanced Fields

> At LASA, a system for teaching a robotic arm how to play minigolf has been developed. The minigolf skill requires two parts: 1) A basic hitting motion model 2) A hitting motion adaptation model. The latter is used to set the parameters of the former such that the robot hits the ball with appropriate hitting angle and hitting speed given the position of the ball. The use of a stereovision system to track the ball enables the robot to solve all challenges involved in playing minigolf autonomously, i.e. tracking the ball, choosing hitting angle and hitting speed, and hitting the ball. This project aims at extending the range of playable fields by improving various aspects of the system. The focus will be on implementing a self-improvement mechanism (to allow the robot to learn from its experience) and investigating alternative models of the hitting motion.

[1] This semester project was co-supervised by Klas Kronander.

Master thesis, *Winter 2010*

Student: Klas Kronander

Title: Learning to Control Planar Hitting Motions of a Robotic Arm in a Mini-Golf-like Task

> This project will acquaint the student with the complexity of learning a control law for a multi-degrees of freedom robot from human demonstrations. We consider a task that mimics some of the difficulties one encounters when playing mini-golf. In such a task, hitting the golf ball with the right orientation and speed is crucial and requires years of training. In this project, the student will further improve and implement tools developed in the laboratory for estimating such control laws. Control laws are expressed as nonlinear autonomous dynamical systems. Estimation is done through nonlinear optimization of Gaussian Mixture Models under stability constraints. The learned model will be implemented both in a dynamic simulator and on a seven degrees of freedom robot arm in a realistic mock-up of a mini-golf terrain.

Semester project, *Spring 2009*

Student: Adrian Arfire

Title: Kinesthetic Teaching of an Acrobatic Maneuver to an Aerial Robot

> Programming by Demonstration (PbD) investigates natural means to teach skills to robots. In this method, the robot learns a given task from a set of demonstrations shown by the user. In our lab, we have widely used this method to teach different dynamical motions (e.g. writing alphabets, putting an object into a container, wiping a tray, etc.) to humanoid robots. In this project, it is desired to extend the applicability of the current framework for aerial robots, and to evaluate its performance for different aerial maneuvers including "loop", "tight turn", "eight maneuvers", "stall", etc. The airplane is a flying wing that is equipped with an IMU-GPS sensor to grab data during the flight. The project divided as follows: 1) investigating a set of parameters from the whole available data from the sensors (e.g. position, velocity, acceleration, control efforts, etc) that can describe well the dynamics of the motion for a specific aerial maneuver, 2) Grabbing data by performing some aerial maneuvers (the flight could be done by a pilot or by the student), 3) Applying the PbD on the data to learn the dynamics, and 4) Improving the efficiency of the controller in accordance to the mission criteria (i.e. the performed maneuver should be as similar as possible to the demonstrations). Another note is that the designed controller should be able to work in conjunction with the aircraft main controller because it takes the control of the vehicle only during the maneuver.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*, 147–169.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*, 716–723.

Aleotti, J., & Caselli, S. (2006). Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems*, *54*, 409–413.

Andersson, R. (1989). Aggressive trajectory generator for a robot ping-pong player. *IEEE Control Systems Magazine*, *9(2)*, 15–21.

Ardizzone, E., Chella, A., & Pirrone, R. (2000). Pose classification using support vector machines. In *proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks* (Vol. 6, pp. 317–322 vol.6). doi: 10.1109/IJCNN.2000.859415

Artstein, Z. (1983). Stabilization with relaxed controls. *Nonlinear Analysis*, *7*, 1163–1173.

Aström, K. J., & Bohlin, T. (1965). Numerical Identification of Linear Dynamic Systems from Normal Operating Records. In *IFAC Symposium on the Theory of Self-adaptive Control Systems.*

Aström, K. J., & Eykhoff, P. (1971, March). System identification-A survey. *Automatica*, *7*(2), 123–162. doi: 10.1016/0005-1098(71)90059-8

Atiya, A. F., Parlos, A. G., Member, S., & Member, S. (2000). New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence. *IEEE Trans. Neural Networks*, *11*, 697–709.

Atkeson, C. G. (1990). Advances in neural information processing systems 2. In D. S. Touretzky (Ed.), (pp. 316–323). Morgan Kaufmann Publishers Inc.

Barbehenn, M., Chen, P., & Hutchinson, S. (1994). An efficient hybrid planner in changing environments. In *IEEE Int. Conf. on Robotics and Automation* (Vol. 4, pp. 2755–2760).

Bartlett, M. S., Littlewort, G., Fasel, I., & Movellan, J. R. (2003). Real Time Face Detection and Facial Expression Recognition: Development and Applications to Human Computer Interaction. In *Conference on Computer Vision and Pattern Recognition Workshop* (Vol. 5).

Bazaraa, M. S., Sherali, H., & Shetty, C. (2006). *Nonlinear programming: Theory and Algorithms* (3rd ed.). John Wiley & Sons.

Benallegue, M., Escande, A., Miossec, S., & Kheddar, A. (2009). Fast C1 Proximity Queries using Support Mapping of Sphere-Torus-Patches Bounding Volumes. In *Proc. IEEE Int. Conf. on Robotics and Automation* (pp. 483–488).

Billard, A., Calinon, S., Dillmann, R., & Schaal, S. (2008). Handbook of Robotics. In (chap. Robot Programming by Demonstration). MIT Press.

Billard, A., & Hayes, G. (1999). DRAMA, a connectionist architecture for control and learning in autonomous robots. *Adaptive Behavior Journal*, *7*(1), 35–64.

Bishop, C. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Clarendon Press.

Bitzer, S., & Vijayakumar, S. (2009). Latent spaces for dynamic movement primitives. In *Proc. 9th IEEE-RAS Int. Conf. Humanoid Robots* (pp. 574–581).

Bonnans, J., & Gilbert, J. (2006). *Numerical Optimization* (2nd ed.). Springer.

Boor, V., Overmars, M., & van der Stappen, A. (1999). The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. Robot. Autom.* (pp. 1018–1023).

Borenstein, J., & Koren, Y. (1991). The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots. *IEEE Trans. on Robotics and Automation*, *7*, 278–88.

Borne, P., & Dieulot, J.-Y. (2005). Fuzzy systems and controllers: Lyapunov tools for a regionwise approach. *Nonlinear Analysis*, 653–665.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Brock, O., & Khatib, O. (1999). Elastic Strips: A framework for integrated planning and execution. In *Proceedings of the International Symposium on Experimental Robotics* (Vol. 250, pp. 328–338). Springer Verlag.

Brock, O., & Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *Int. Journal of Robotics Research*, *21*(12), 1031–1052.

Brock, O., Kuffner, J., & Xiao, J. (2007). Motion for Manipulation Tasks [incollection]. In *Handbook of Robotics*. Springer. doi: 10.1007/978-3-540-30301-5_6

Bryson, A. E., & Ho, Y. (1975). *Applied Optimal Control: Optimization, Estimation, & Control*. Taylor & Francis; Revised edition.

Bullock, D., Bongers, R. M., Lankhorst, M., & Beek, P. J. (1999). A vector-integration-to-endpoint model for performance of viapoint movements. *Neural Networks*, *12*(1), 1–29. doi: 10.1016/S0893-6080(98)00109-9

Bullock, D., & Grossberg, S. (1988a). Neural dynamics of planned arm movements: Emergent invariants and speed-accuracy properties during trajectory formation. *Psychological Review*, *95*, 49–90.

Bullock, D., & Grossberg, S. (1988b). The VITE Model: A Neural Command Circuit for Generating Arm and Articulator Trajectories. *Dynamic Patterns in Complex Systems*, 305–326.

Burns, B., & Brock, O. (2005). Toward Optimal Configuration Space Sampling. In *Proc. of Robotics: Science and Systems.*

Butz, M., Herbort, O., & Hoffmann, J. (2007). Exploiting redundancy for flexible behavior: Unsupervised learning in a modular sensorimotor control architecture. *Psychological Review*, *114*, 1015–1046.

Calinon, S. (2009). *Robot Programming by Demonstration: A Probabilistic Approach.* EPFL/CRC Press.

Calinon, S., & Billard, A. (2005). Recognition and Reproduction of Gestures using a Probabilistic Framework combining PCA, ICA and HMM. In *proceedings of the International Conference on Machine Learning (ICML)* (pp. 105–112).

Calinon, S., & Billard, A. (2007a). Active Teaching in Robot Programming by Demonstration. In *proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* (pp. 702–707).

Calinon, S., & Billard, A. (2007b). Incremental Learning of Gestures by Imitation in a Humanoid Robot. In *proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)* (pp. 255–262).

Calinon, S., D'halluin, F., Sauser, E., Caldwell, D., & Billard, A. (2010). Learning and reproduction of gestures by imitation: An approach based on Hidden Markov Model and Gaussian Mixture Regression. *IEEE Robotics and Automation Magazine*, *17:2*, 44–54.

Calinon, S., Guenter, F., & Billard, A. (2007). On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE transactions on systems, man and cybernetics*, *37*(2), 286–298.

Calinon, S., Pistillo, A., & Caldwell, D. G. (2011). Encoding the time and space constraints of a task in explicit-duration hidden Markov model. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).*

Calinon, S., Sauser, E., Billard, A., & Caldwell, D. (2010). Evaluation of a probabilistic approach to learn and reproduce gestures by imitation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 2671–2676).

Canny, J. (1988). *The Complexity of Robot Motion Planning.* The MIT Press.

Choi, W., & Latombe, J.-C. (1991, nov). A reactive architecture for planning and executing robot motions with incomplete knowledge. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems* (pp. 24–29).

Coates, A., Abbeel, P., & Ng., A. Y. (2008). Learning for Control from Multiple Demonstrations. In *Proc. 25th Int. Conf. on Machine Learning (ICML)* (pp. 144–151).

Conner, D. C., Choset, H., & Rizzi, A. (2006, August). Integrated Planning and Control for Convex-bodied Nonholonomic Systems Using Local Feedback. In *proceedings of Robotics: Science and Systems II* (pp. 57–64). Philadelphia, PA: MIT Press.

Conner, D. C., Rizzi, A. A., & Choset., H. (2003). Composition of local potential functions for global robot control and navigation. In *proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3546–3551).

Connolly, C., Burns, J., & Weiss, R. (1990). Path planning using Laplace's equation. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)* (pp. 2102–2106).

Cypher, A. (1993). *Watch what I do: programming by demonstration* (A. Cypher et al., Eds.). Cambridge, MA, USA: MIT Press.

Dautenhahn, K., & Nehaniv, C. L. (2002). The agent-based perspective on imitation. In (pp. 1–40). Cambridge, MA, USA: MIT Press.

Davis, L. (1987). *Genetic Algorithms and Simulated Annealing (Research Notes in Artificial Intelligence)*. Morgan Kaufmann Publishers Inc.

Delcomyn, F. (1980). Neural basis for rhythmic behaviour in animals. *Science*, *210*, 492–498.

Delson, N., & West, H. (1996). Robot Programming by Human Demonstration: Adaptation and Inconsistency in Constrained Motion,. In *Proc. IEEE International Conference on Robotics and Automation* (pp. 30–36).

Dempster, A., & Rubin, N. L. D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*(1), 1–38.

Diankov, R., & Kuffner, J. (2007). Randomized Statistical Path Planning. In *Proc. of IEEE/RSJ Int. Conf. on Robots and Systems (IROS)* (pp. 1–6).

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Dixon, K., & Khosla, P. (2004a). Trajectory Representation Using Sequenced Linear Dynamical Systems. In *IEEE International Conference on Robotics and Automation.*

Dixon, K., & Khosla, P. (2004b, 26-may 1,). Trajectory representation using sequenced linear dynamical systems. In *proceedings of the IEEE International Conference on Robotics and Automation* (Vol. 4, pp. 3925–3930 Vol.4).

Donald, B., Xavier, P., Canny, J., & Reif, J. (1993). Kinodynamic planning. *Journal of ACM*, *40*, 1048–1066.

Eberhart, R. C., Shi, Y., & Kennedy, J. (2001). *Swarm Intelligence* (1st ed.). Morgan Kaufmann.

Feder, H., & Slotine, J.-J. (1997). Real-time path planning using harmonic potentials in dynamic environments. In *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)* (Vol. 1, pp. 874–881).

Ferbach, P. (1996). A method of progressive constraints for nonholonomic motion planning. In *IEEE Int. Conf. Robot. Autom.* (pp. 2949–2955).

Fraichard, T., Hassoun, M., & Laugier, C. (1991). Reactive motion planning in a dynamic world. In *Proc. of the IEEE Int. Conf. on Advanced Robotics* (pp. 1028–1032).

Gaudiano, P., & Grossberg, S. (1992). Adaptive vector integration to endpoint: Self-organizing neural circuits for control of plannedmovement trajectories. *Human Movement Science*, *11*, 141–155.

Gevers, M. (2006). A personal view of the development of system identification: A 30-year journey through an exciting field. *Control Systems, IEEE*, *26*(6), 93–105. doi: 10.1109/MCS.2006.252834

Gilbert, G. T. (1991). Positive definite matrices and Sylvester's criterion. *The American Mathematical Monthly (Mathematical Association of America)*, *98(1)*, 44–46.

Go, J., Vu, T., & Kuffner, J. (2004). Autonomous Behaviors for Interactive Vehicle Animations. In *Proc. ACM SIGGRAPH Symposium on Computer Animation (SCA 2004).* ACM.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co.

Gribovskaya, E., Khansari-Zadeh, S. M., & Billard, A. (2010). Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators. *The International Journal of Robotics Research*, *30*, 1–37.

Grillner, S. (1975). Locomotion in Vertebrates: Central Mechanisms and Reflex Intraction. *Physiol Rev.*, *55*, 247–304.

Grillner, S. (1985). Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, *228*, 143–149.

Grollman, D., & Billard, A. (2011). Donut as I do: Learning from Failed Demonstrations. In *proceedings of IEEE International Conference on Robotics and Automation.*

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. doi: 10.1109/TSSC.1968.300136

Hersch, M., Guenter, F., Calinon, S., & Billard, A. (2008). Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations. *IEEE Transactions on Robotics*, 1463–1467.

Hinton, G. E. (2007). Boltzmann machine. *Scholarpedia*, *2*(5), 1668.

Ho, B., & Kalman, R. (1966). Effective construction of linear state-variable models from input-output functions. *Regelungstechnik*, *12*, 545–548.

Hoffmann, H., Pastor, P., Park, D.-H., & Schaal, S. (2009). Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *Proc. of Int. Conf. on Robotics and Automation* (pp. 2587–2592).

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *proceedings of National Academy of Sciences* (Vol. 79, pp. 2554–2558).

Hopfield, J. J. (2007). Hopfield network. *Scholarpedia*, *2*(5), 1977. doi: 10.4249/scholarpedia.1977

Hovland, G., Sikka, P., & McCarragher, B. (1996). Skill acquisition from human demonstration using a hidden Markov model. In *IEEE International Conference on Robotics and Automation* (Vol. 3, pp. 2706–2711).

Howard, M., Klanke, S., Gienger, M., Goerick, C., & Vijayakumar, S. (2009). A novel method for learning policies from variable constraint data. *Autonomous Robots*, *27*, 105-121.

Hsu, D., Kindel, R., Latombe, J., & Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, *21*, 233–255.

Hsu, D., Latombe, J., & Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, *25*(7).

Hsu, J., & Meyer, A. (1968). *Modern Control Principles and Applications.* McGraw-Hill.

Hwang, J.-H., Arkin, R., & Kwon, D.-S. (2003). Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In *proceedings of the IEEE/RSJ IROS* (Vol. 2, pp. 1444–1449).

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, *21*(4), 642–653. doi: 10.1016/j.neunet.2008.03.014

Ijspeert, A. J., Hallam, J., & Willshaw, D. (1998). Evolution of a central pattern generator for the swimming and trotting gaits of the salamander. In *proceedings of the Third International Conference on Computational Intelligence & Neurosciences, (ICCIN98).*

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002a). Learning Attractor Landscapes for Learning Motor Primitives. In *NIPS* (pp. 1523–1530).

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002b). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA)* (pp. 1398–1403).

Ijspeert, A. J., Nakanishi, J., Shibata, T., & Schaal, S. (2001). Nonlinear dynamical systems for imitation with humanoid robots. In *IEEE-RAS International Conference on Humanoid Robots.*

Inamura, T., Toshima, I., & Nakamura, Y. (2002). Acquiring Motion Elements for Bidirectional Computation of Motion Recognition and Generation. In *ISER* (pp. 372–381). Springer.

Ioannou, P. A., & Sun, J. (1996). *Robust Adaptive Control.* Prentice Hall.

Iossifidis, I., & Schöner, G. (2006). Dynamical Systems Approach for the Autonomous Avoidance of Obstacles and Joint-limits for a Redundant Robot Arm. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)* (pp. 580–585).

Ito, M., & Tani, J. (2004). On-line Imitative Interaction with a Humanoid Robot Using a Dynamic Neural Network Model of a Mirror System. *Adaptive Behavior*, *12*(2), 93–115.

Jaeger, H., Lukoevicius, M., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*, *20*(3), 335–352.

Jiang, Z., Lin, Y., & Wang, Y. (2009). Stabilization of nonlinear time-varying systems: a control lyapunov function approach. *Journal of Systems Science and Complexity*, *22*, 683–696. (10.1007/s11424-009-9195-1)

Jordan, M., & Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science: A Multidisciplinary Journal*, *16*, 307–354.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894. doi: 10.1177/0278364911406761

Kavraki, L. E., & LaValle, S. M. (2007). Motion Planning [incollection]. In *Handbook of Robotics.* Springer. doi: 10.1007/978-3-540-30301-5_6

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, *12*(4), 566–580.

Kelley, C. T. (1999). Line Search Methods and the Armijo Rule. In *Iterative Methods for Optimization* (pp. 40–52). SIAM.

Kelso, J. (1995). *Dynamic patterns: The self-organization of brain and behavior.* Cambridge, MA: MIT Press.

Khalil, H. (1996). *Nonlinear systems.* Prentice Hall Upper Saddle River, NJ.

Khansari-Zadeh, S. M., & Billard, A. (2010a). BM: An iterative algorithm to learn stable non-linear dynamical systems with Gaussian mixture models. In *Proceeding of the International Conference on Robotics and Automation (ICRA)* (pp. 2381–2388).

Khansari-Zadeh, S. M., & Billard, A. (2010b). Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (pp. 2676–2683).

Khansari-Zadeh, S. M., & Billard, A. (2011). Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Trans. on Robotics*, *27*(5), 943–957. doi: 10.1109/TRO.2011.2159412

Khansari-Zadeh, S. M., & Billard, A. (2012). A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, *32*, 433–454.

Khansari-Zadeh, S. M., Kronander, K., & Billard, A. (2012). Learning to play minigolf: A dynamical system-based approach. *Advanced Robotics*, 1-27. doi: 10.1080/01691864.2012.728692

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, *5*, 90–98.

Kim, J.-O., & Khosla, P. (1992). Real-time obstacle avoidance using harmonic potential functions. *IEEE Trans. on Robotics and Automation*, *8*(3), 338–349.

Kim, S., & Billard, A. (2012). Estimating the non-linear dynamics of free-flying objects. *Robotics and Autonomous Systems*, *60*(9), 1108–1122. doi: 10.1016/j.robot.2012.05.022

Kim, S., Gribovskaya, E., & Billard, A. (2010). Learning Motion Dynamics to Catch a Moving Object. In *the 10th IEEE-RAS International Conference on Humanoid Robots* (pp. 106–111).

Kober, J., Mulling, K., Kromer, O., Lampert, C. H., Scholkopf, B., & Peters, J. (2010). Movement Templates for Learning of Hitting and Batting. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 853–858).

Kober, J., Oztop, E., & Peters, J. (2010). Reinforcement Learning to adjust Robot Movements to New Situations,. In *Proc. of Robotics: Science and Systems (RSS).*

Koditschek, D. (1987). Exact robot navigation by means of potential functions: Some topological considerations. In *proceedings of the IEEE International Conference on Robotics and Automation* (Vol. 4, pp. 1–6).

Kokotovic, P., & Arcak, M. (2001). Constructive nonlinear control: a historical perspective. *Automatica*, *37*, 637–662.

Kronander, K., Khansari Zadeh, S. M., & Billard, A. (2011). Learning to control planar hitting motions in a minigolf-like task. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (pp. 710–717).

Kuffner, J., & LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proc. of IEEE Int. Conf. on Robotics and Automation* (Vol. 2, pp. 995–1001).

Kulchenko, P., & Todorov, E. (2011). First-exit model predictive control of fast discontinuous dynamics: Application to ball bouncing. In *IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 2144–2151).

Kulic, D., Takano, W., & Nakamura, Y. (2008). Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *The Int. Journal of Robotics Research*, *27*(7), 761–784.

Kulvicius, T., Ning, K., Tamosiunaite, M., & Worgotter, F. (2012). Joining Movement Sequences: Modified Dynamic Movement Primitives for Robotics Applications Exemplified on Handwriting. *IEEE Transactions on Robotics*, *28*(1), 145–157. doi: 10.1109/TRO.2011.2163863

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1989). Teaching by showing: Generating robot programs by visual observation of human performance. In *International Symposium of Industrial Robots* (pp. 119–126).

Kutner, M. H., Nachtsheim, C., & Neter, J. (2005). *Applied linear regression models*. McGraw-Hill/Irwin.

Lahanas, M., Kemmerer, T., Milickovic, N., Karouzakis, K., Baltas, D., & Zamboglou, N. (2000). Optimized Bounding Boxes for Three-dimensional Treatment Planning in Brachytherapy. *Medical Physics*, *27*(10), 2333–2342.

LaValle, S., Branicky, M., & Lindemann, S. (2004). On the relationship between classical grid search and probabilistic roadmaps. *Int. J. Robot. Res.*, *23*, 673–692.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Lee, H. K. H. (2004). *Bayesian Nonparametrics Via Neural Networks*. Cambridge University Press.

Li, Y., Gong, S., & Liddell, H. (2000). Support vector regression and classification based multi-view face detection and recognition. In *proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition* (pp. 300–305).

Lieberman, H. (2001). *Your wish is my command: programming by example*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Lien, J., Thomas, S., & Amato, N. (2003). A general framework for sampling on the medial axis of the free space. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Lin, D.-T., Dayhoff, J. E., & Ligomenides, P. A. (1995). Trajectory production with the adaptive time-delay neural network. *Neural Networks*, *8*(3), 447–461.

Lindemann, S. R., & LaValle, S. M. (2005). Smoothly blending vector fields for global robot navigation. In *proceedings of the IEEE Conference Decision & Control* (pp. 3353–3559).

Ljung, L. (1999). *System Identification - Theory For the User* (2nd ed.). Prentice Hall, Upper Saddle River, N.J.

Ljung, L. (2010). Perspectives on system identification. *Annual Reviews in Control*, *34*(1), 1–12. doi: 10.1016/j.arcontrol.2009.12.001

Lohmiller, W., & Slotine, J.-J. E. (1998). On Contraction Analysis for Nonlinear Systems. *Automatica*, *34*(6), 683–696.

Lozano-Perez, T. (1983). Robot programming. In *proceedings of the IEEE* (Vol. 71, pp. 821–841).

Lozano-Perez, T., & Wesley, M. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*, 560–570.

Lukosevicius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149.

Lumelsky, V., & Skewis, T. (1990). Incorporating Range Sensing in the Robot Navigation Function. *IEEE Trans. on Systems, Man, and Cybernetics*, *20*, 1058–1069.

Maass, W., Natschlager, T., & Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Maciejewski, A. A., & Klein, C. A. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *Int. Journal of Robotics Research*, *4*, 109–116.

Marder, E., & Bucher, D. (2001). Central pattern generators and the control of rhythmic movements. *Current Biology*, *11*, 986–996.

McLachlan, G., & Peel, D. (2000). *Finite Mixture Models*. Wiley.

Medsker, L., & Jain, L. (Eds.). (1999). *Recurrent Neural Networks Design and Applications*. CRC Press.

Mendel, J. M., & McLaren, R. W. (1970). Reinforcement learning control and pattern recognition systems. *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, 287–318. (Academic Press, New York)

Milne-Thomson, L. M. (1960). *Theoretical Hydrodynanamics* (4th ed.). The Macmillan Company.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., & Zabowski, D. (1994). Experience With a Learning Personal Assistant. *Communications of the ACM*, *37*(7), 80–91.

Mülling, K., Kober, J., & Peters, J. (2011). A biomimetic approach to robot table tennis. *Adaptive Behavior*, *9*(5), 359–376.

Münch, S., Kreuziger, J., Kaiser, M., & Dillmann, R. (1994). Robot Programming by Demonstration (RPD) - Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Robot Programming Systems. In *Proc. of the 24th Int. Symposium on Industrial Robots* (pp. 685–693).

Muehlig, M., Gienger, M., Hellbach, S., Steil, J., & Goerick, C. (2009). Task level imitation learning using variance-based movement optimization. In *IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 1177–1184).

Myers, C., & Rabiner, L. (1981). A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, *60*(7), 1389–1409.

Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., & Kawato, M. (2004). Learning from demonstration and adaptation of biped locomotion. *Robot. Auton. Syst.*, *47*, 79–91.

Nemec, B., Tamosiunaite, M., Worgotter, F., & Ude, A. (2009). Task adaptation through exploration and action sequencing. In *Humanoids 2009* (pp. 610–616).

Nguyen-Tuong, D., Seeger, M., & Peters, J. (2008). Computed torque control with nonparametric regression models. In *American Control Conference* (pp. 212–217).

Ogata, K. (2001). *Modern Control Engineering* (4th ed.). Prentice Hall.

Ogawara, K., Takamatsu, J., Kimura, H., & Ikeuchi, K. (2003). Extraction of essential interactions through multiple observations of human demonstrations. *IEEE Trans. Indust. Electron.*, *50*(4), 667–675.

Osuna, E., Freund, R., & Girosit, F. (1997, jun). Training support vector machines: an application to face detection. In *proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997* (pp. 130–136).

Pardowitz, M., Knoop, S., Dillmann, R., & Zollner, R. (2007). Incremental Learning of Tasks From User Demonstrations, Past Experiences, and Vocal Comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *37*(2), 322–332.

Park, D.-H., Hoffmann, H., Pastor, P., & Schaal, S. (2008). Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Proc. of the IEEE Int. Conf. on Humanoid Robotics* (pp. 91–98).

Parrilo, P. A. (2000). *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. Unpublished doctoral dissertation, California Institute of Technology.

Pastor, P., Hoffmann, H., Asfour, T., & Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 1293–1298).

Pearlmutter, B. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, *1*, 263–269.

Pearlmutter, B. A. (1995). Gradient calculation for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks*.

Perk, B. E., & Slotine, J.-J. E. (2006). Motion Primitives for Robotic Flight Control. *CoRR*, *abs/cs/0609140*.

Peters, J., & Schaal, S. (2008). Learning Robot Dynamics for Computed Torque Control Using Local Gaussian Processes Regression. In *proceedings of the 2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems* (pp. 59–64).

Pettersson, S., & Lennartson, B. (1997). Exponential stability analysis of nonlinear systems using LMIs. In *proceedings of the 36th IEEE Conference on Decision and Control*.

Pineda, F. J. (1987). Generalization of backpropagation to recurrent neural networks. *Physics Review Letters*, *18*, 2229–2232.

Primbs, J. A., Nevistic, V., & Doyle, J. C. (1999). Nonlinear Optimal Control: A Control Lyapunov Function and Receding Horizon Perspective. *Asian Journal of Control*, *1*(1), 14–24.

Quinlan, S., & Khatib, O. (1993). Elastic bands: connecting path planning and control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (Vol. 2, pp. 802–807).

Raibert, M. (1986). *Legged robots that balance*. Cambridge MA: MIT Press.

Ramanantsoa, M., & Durey, A. (1994). Towards a stroke construction model. *International Journal of Table Tennis Science*, *2*, 97–114.

Rasmussen, C., & Williams, C. (2006). *Gaussian processes for machine learning*. Springer.

Reinhart, R. F., & Steil, J. J. (2011). Neural learning and dynamical selection of redundant solutions for inverse kinematic control. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)* (pp. 564–569).

Righetti, L., Buchli, J., & Ijspeert, A. J. (2006). Dynamic hebbian learning in adaptive frequency oscillators. *Physica D*, *216*, 269–281.

Rimon, E., & Koditschek, D. (1992, oct). Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on*, *8*(5), 501–518. doi: 10.1109/70.163777

Rochat, S. D., Gay, L. R. S., & Ijspeert, A. J. (2011). Towards simple control for complex, autonomous robotic applications: Combining discrete and rhythmic motor primitives. *Autonomous Robots*, *31*(2), 155–181.

Salle, J. L., & Lefschetz, S. (1961). *Stability by Lyapunov direct method*. New York: Academic Press.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, *3*(6), 233–242.

Schaal, S. (2003). Movement planning and imitation by shaping nonlinear attractors. In *proceedings of the 12th yale workshop on adaptive and learning systems.*

Schaal, S., & Atkeson, C. (1994). Robot juggling: implementation of memory-based learning. *IEEE Control Systems Magazine*, *14*(1), 57–71.

Schaal, S., Atkeson, C., & Vijayakumar, S. (2002). Scalable Locally Weighted Statistical Techniques for real time robot learning. *Applied Intelligence - Special issue on Scalable Robotic Applications of Neural Networks*, *17*(1).

Schaal, S., Ijspeert, A. J., & Billard, A. (2003). Computational Approaches to Motor Learning by Imitation. *Philosophical Transactions: Biological Sciences (The Royal Society)*(1431), 537–547.

Schaal, S., Kotosaka, S., & Sternad, D. (2000). Nonlinear dynamical systems as movement primitives. In *proceedings of the IEEE-RAS International Conference on Humanoid Robots.*

Schaal, S., Peters, J., Nakanishi, J., & Ijspeert, A. J. (2003). Control, Planning, Learning, and Imitation with Dynamic Movement Primitives. In *Workshop on bilateral paradigms on humans and humanoids, IEEE International Conference on Intelligent Robots and Systems.*

Schaal, S., Peters, J., Nakanishi, J., & Ijspeert, A. J. (2004). Learning movement primitives. In *Int. symposium on robotics research (ISRR2003)* (pp. 561–572). Springer.

Schwartz, J., & Sharir, M. (1983). On the "piano movers" problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, *4*, 298–351.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *6*, 461–464.

Sciavicco, L., & Siciliano, B. (2000). *Modelling and control of robot manipulators*. Springer.

Söderström, T., & Stoica, P. (1989). *System Identification*. Prentice Hall.

Segre, A., & DeJong, G. (1985). Explanation-based manipulator learning Acquisition of planning ability through observation. In *IEEE Conference on Robotics and Automation (ICRA)* (pp. 555–560).

Selverston, A. I. (1980). Are central pattern generators understandable? *The Behavioral and Brain Sciences*, *3*, 555–571.

Shilane, P., Min, P., Kazhdan, M., & Funkhouser, T. (2004). The Princeton Shape Benchmark. In *Shape Modeling International.*

Shukla, A., & Billard, A. (2012a). Augmented-SVM: Automatic space partitioning for combining multiple non-linear dynamics. In *Advances in Neural Information Processing Systems (NIPS)*.

Shukla, A., & Billard, A. (2012b). Coupled dynamical system based arm-hand grasping model for learning fast adaptation strategies. *Robotics and Autonomous Systems*, *60*(3), 424–440. doi: 10.1016/j.robot.2011.07.023

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modelling, Planning and Control*. Springer.

Sigaud, O., Salaün, C., & Padois, V. (2011). On-line regression algorithms for learning mechanical models of robots: A survey. *Robotics and Autonomous Systems*, *59*(12), 1115–1129.

Simmons, R. (1996). The curvature-velocity method for local obstacle avoidance. In *Proc. of the IEEE Int. Conf. on Robotics and Automation* (Vol. 4, pp. 3375–3382).

Slotine, J., & Li, W. (1991). *Applied Nonlinear Control*. Prentice-Hall.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, *14*(3), 199–222.

Sontag, E. D. (1998). *Mathematical Control Theory: Deterministic Finite Dimensional Systems* (2nd ed.). Springer, New York.

Sontag, E. D. (2008). Input to State Stability: Basic Concepts and Results. In *Nonlinear and Optimal Control Theory* (pp. 163–220). Springer Berlin/Heidelberg.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & van der Linde, A. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, *64*, 583–639.

Sprunk, C., Lau, B., Pfaffz, P., & Burgard, W. (2011). Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)* (pp. 72–77).

Strandbergtrees, M. (2004). Augmenting RRT-planners with local trees. In *IEEE Int. Conf. Robot. Autom.* (pp. 3258–3262).

Sudareshan, M., & Condarcure, T. (1998). Recurrent neural-network training by a learning automaton approach for trajectory learning and control system design. *IEEE Transactions on Neural Networks*, *9*(3), 354–368.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Taga, G., Yamaguchi, Y., & Shimizu, H. (1991). Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, *65*, 147–159.

Toussaint, M. (2009). Robot Trajectory Optimization using Approximate Inference. In *Proc. 25th Int. Conf. on Machine Learning (ICML)* (pp. 1049–1056).

Tso, S., & Liu, K. (1996). Hidden Markov model for intelligent extraction of robot trajectory command from demonstrated trajectories. In *Proc. IEEE International Conference on Industrial Technology (ICIT)* (pp. 294–298).

Ude, A. (1993). Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, *11*(2), 113–127.

Ude, A., Gams, A., Asfour, T., & Morimoto, J. (2010). Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. *IEEE Trans. on Robotics*, *26*(5), 800–815.

Udupa, S. (1977). *Collision detection and avoidance in computer controlled manipulators*. Unpublished doctoral dissertation, Dept. of Electical Engineering, California Institute of Technology.

Vannoy, J., & Xiao, J. (2008). Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes. *IEEE Trans. on Robotics*, *24*, 1199–1212.

Vijayakumar, S., D'Souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, *17*(12), 2602–2634.

Vijayakumar, S., & Schaal, S. (2000). Locally Weighted Projection Regression: An O(n) algorithm for incremental real time learning in high dimensional space. In *Proc. of 17th Int. Conf. on Machine Learning (ICML)* (pp. 1079–1086).

Waldherr, S., Thrun, S., & Romero, R. (2000). A Gesture-Based Interface for Human-Robot Interaction. *Autonomous Robots*, *9*(2), 151–173.

Waltz, M., & Fu, K. (1965). A heuristic approach to reinforcement learning control systems. *IEEE Trans. on Automatic Control*, *10*, 390–398.

Waltz, R. A., Morales, J. L., Nocedal, J., & Orban, D. (2006). An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, *107*(3), 391–408.

Waydo, S., & Murray, R. (2003). Vehicle motion planning using stream functions. In *Proc. of the IEEE Int. Conf. on the Robotics and Automation (ICRA)* (Vol. 2, pp. 2484–2491).

Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). In H. Maurer (Ed.), *New Results and New Trends in Computer Science* (Vol. 555, pp. 359–370). Springer Berlin / Heidelberg.

Wolpert, D., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, *11*, 1317–1329.

Yamane, K., Kuffner, J. J., & Hodgins, J. K. (2004). Synthesizing Animations of Human Manipulation Tasks. *ACM Transactions on Graphics (SIGGRAPH 2004)*, *23*(3).

Yang, J., Xu, Y., & Chen, C. (1997). Human Action Learning via Hidden Markov Model. *IEEE Trans. Syst. Man Cybernet*, *27*(1), 34–44.

Yang, L., & LaValle, S. (2004). The sampling-based neighborhood graph: an approach to computing and executing feedback motion strategies. *IEEE Transactions on Robotics and Automation*, *20*(3), 419–432.

Yang, Y., & Brock, O. (2007). Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In *Proc. Robotics: Science and Systems (RSS)*.

Yershova, A., Jaillet, L., Simeon, T., & LaValle, S. (2005). Dynamic-domain RRTs: efficient exploration by controlling the sampling domain. In *Proc. of the IEEE Int. Conf. on the Robotics and Automation (ICRA).*

Yoshida, E., & Kanehiro, F. (2011). Reactive robot motion using path replanning and deformation. In *Proc. IEEE Int. Conf. on Robotics and Automation* (pp. 5457–5462).

Zadeh, L. (1956). On the identification problem. *IRE Transactions on Circuit Theory*, *3*, 277–281.

Zollner, R., Rogalla, O., Dillmann, R., & Zollner, M. (2002). Understanding users intention: programming fine manipulation tasks by demonstration. In *proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (Vol. 2, pp. 1114–1119).

Zucker, M., Kuffner, J., & Branicky, M. (2007). Multiple RRTs for rapid replanning in dynamic environments. In *IEEE Conference on Robotics and Automation.*

S. Mohammad Khansari Zadeh

17 September 1983
Iranian Citizen
Married

Av. de Florissant, 34
1020 Renens, Switzerland
+41 78 8886041
mohammad.khansari@epfl.ch
http://www.khansari.org

Research Interests

- Modeling and Control of Movement Primitives
- Imitation Learning
- Dynamical Systems
- Nonlinear Control

Education

2008–2012 **PhD Student in Robotics**, *Learning Algorithms and Systems Laboratory (LASA) at the Swiss Federal Institute of Technology Lausanne (EPFL)*, Switzerland
Thesis title: A Dynamical System-based Approach to Modeling Stable Robot Control Policies via Imitation Learning
Thesis advisor: Prof. Aude Billard.

2005–2008 **MSc in Aerospace Engineering (specialization in Flight Dynamics and Control)**, *Aerospace Engineering Department*, Sharif University of Technology, Tehran, Iran.

2001–2005 **BSc in Aerospace Engineering**, *Aerospace Engineering Department*, Sharif University of Technology, Tehran, Iran.

Professional & Research Experience

2008–present **Research Assistant**, *LASA, EPFL*, Lausanne, Switzerland.

- Teaching assistant for the course *"Practical Robots"*
- Supervision of master theses and projects

2008–present **Affiliated European Projects:**.

- AMARSi (Adaptive Modular Architectures for Rich Motor Skills), `http://www.amarsi-project.eu`
- Robot@CWE (Robots in the future collaborative working environments), `http://robot-at-cwe.eu`

2011 **German Aerospace Research center (DLR)**, *Summer school on "Impedance Control"*, Frauenchiemsee, Germany.

2005–2008 **Teaching Assistant**, *Sharif University of Technology*, Tehran, Iran.

- Graduate course *"Modeling and Simulation of Aerospace Vehicle Dynamics"*
- Undergraduate course *"Flight Dynamics II"*
- Undergraduate course *"Computer Aided Design (CAD)"*

2005–2012 **Modeling & Controller Design**, Extensive modeling and controller development in C++, MATALB and Simulink for a wide variety of vehicles, including various degrees of freedom robot arms and unmanned aerial vehicles.

2004–2005 **Head of Aerospace Students Community**, *Sharif University of Technology*, Iran.

2003–2005 **Editor-in-Chief of Scientific Journal of Aerospace Students (Owj)**, *Sharif University of Technology*, Tehran, Iran.

Awards and Honors

2011 **JTCF Novel Technology Paper Award for Amusement Culture**, *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, San Francisco.

2005–2007 **Ranked 1st among graduate Aerospace Engineering students (majoring in Flight Dynamics and Control)**, *Sharif University of Technology*.

2005–2007 **Fellowship**, *Free M.Sc. education*, Sharif University of Technology.

2005 **Direct admission to M.Sc. program (without doing the compulsory entrance exam)**, *Aerospace Engineering Department*, Sharif University of Technology.

2001–2005 **Ranked 1st among undergraduate Aerospace Engineering students**, *Sharif University of Technology*.

2001–2005 **Fellowship**, *Free undergraduate education*, Sharif University of Technology.

2004 **The eligible head of Aerospace Students Community**, *Awarded by the dean of Extra Curriculum and the dean of Aerospace Engineering Department, Sharif Uni. of Technology*.

Publications

Journals

Learning to Play Minigolf: A Dynamical System-based Approach
S.M. Khansari Zadeh, K. Kronander, and A. Billard
Advanced Robotics, p. 1–27, 2012.

A Dynamical System Approach to Realtime Obstacle Avoidance
S.M. Khansari Zadeh and A. Billard
Autonomous Robots, 32(4), p. 433–454, 2012.

Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models
S.M. Khansari Zadeh and A. Billard
IEEE Transaction on Robotics, 27(5), p. 943–957, 2011.

Vision-Based Navigation in Auto. Close Proximity Operations Using Neural Networks
S.M. Khansari Zadeh and F. Saghafi
IEEE Transactions on Aerospace and Electronic Systems, 47(2), p. 864–883, 2011.

Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators
E. Gribovskaya, S.M. Khansari Zadeh, and A. Billard
International Journal of Robotics Research, 30(1), p. 80–117, 2010.

Aircraft Visual Identification by Neural Networks
F. Saghafi, S. M. Khansari Zadeh and V. Etiminanbakhsh
International Journal of Aerospace Science and Technology (JAST), 5(3), p. 123–128, 2008.

Conference Proceedings

Learning to Control Planar Hitting Motions in a Minigolf-like Task
K. Kronander, S.M. Khansari Zadeh, and A. Billard
In proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), p. 710–717, 2011, **Received the JTSC Novel Technology Paper Award**.

Imitation learning of Globally Stable Non-Linear Point-to-Point Robot Motions using Nonlinear Programming
S.M. Khansari Zadeh and A. Billard
In proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, p. 2676–2683, 2010.

BM: An Iterative Method to Learn Stable Non-Linear Dynamical Systems with Gaussian Mixture Models
S.M. Khansari Zadeh and A. Billard
In proc. of the Int. Conf. on Robotics and Automation (ICRA), p. 2381–2388, 2010.

Intelligent Landing of Autonomous Aerial Vehicles Using Fuzzy Logic Control
F. Saghafi, S. Pouya, and S.M. Khansari Zadeh
IEEE Aerospace Conference, 2009.

Vision-Based Trajectory Tracking Controller for Auto. Close Proximity Operations
F. Saghafi and S.M. Khansari Zadeh
IEEE Aerospace Conference, 2008.

Conference Workshops

Realtime Avoidance of Fast Moving Objects: A Dynamical System-based Approach
S.M. Khansari Zadeh and A. Billard
In electronic proc. of the Workshop on Robot Motion Planning: Online, Reactive, and in Real-Time, The IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2012.

Learning to Play Mini-Golf from Human Demonstration using Autonomous Dynamical Systems
S.M. Khansari Zadeh and A. Billard
In electronic proc. of the Workshop on New Developments in Imitation Learning, Int. Conf. on Machine Learning (ICML), 2011.

Learning and Control of UAV maneuvers Based on Demonstrations
S.M. Khansari Zadeh and A. Billard
Presented in the Workshop on Autonomous Flying vehicles: Fundamentals and Applications, Robotics: Science and Systems (RSS), 2009.

Theses

Vision-Based Controller for Autonomous Close Proximity Operations *[in Farsi]*
S.M. Khansari Zadeh, MSc Thesis, Aerospace Engineering Department, Sharif University of Technology, 2008

Design of an Optimal Tracking Controller for Cruise Flight *[in Farsi]*
S.M. Khansari Zadeh, BSc Thesis, Aerospace Engineering Department, Sharif University of Technology, 2005

Academic Activities

2012 **Program Committee**, *AAAI Fall Symposium on "Robots Learning Interactively from Human Teachers"*, Arlington, USA.

2012 **External Expert**, *at the exam "Models of biological sensory-motor systems" (graduate course)*, EPFL, 2012.

2011 **Program Committee**, *RSS Workshop on "The State of Imitation Learning"*, Los Angeles, USA.

2010 **Panel Member**, *AAAI Spring Symposium on "Educational Robotics and Beyond: Design and Evaluation"*, Palo Alto, USA.

2010–present **Journal Reviewer**, *IEEE Transaction on Robotics, Autonomous Robots.*

2010–present **Conference Reviewer**, *ICRA, IROS, Humanoids, HRI.*

Languages

English Fluent
French Intermediate
Farsi Mother Tongue

Technical skills

Robots Barrett WAM arm, KUKA DLR arm, Humanoid iCub, Katana arm, Humanoid HOAP-3
Programming MATLAB & Simulink; C++, Qt, ROS, Python, Parallel Programming (MPI)
Modeling Solid Works, AutoCAD
Graphics Photoshop, Illustrator
O.S. Windows, Linux, Mac OS X
Misc. Latex, Microsoft Office

Hobbies and Interests

Ski, Tennis, Swimming, Volleyball, Hiking